AGNO: An Adaptive Group Communication Scheme for Unstructured P2P Networks*

Dimitrios Tsoumakos and Nick Roussopoulos

Department of Computer Science
University of Maryland, College Park
{dtsouma,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. Our method achieves high-success content transmission at a cost at least two times smaller than proposed techniques for unstructured networks.

1 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., [1–6]).

A number of methods have been proposed to implement multicast communication utilizing some popular P2P overlays, e.g., [3, 7–9]. 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 existing 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, many popular P2P applications operate on *unstructured* networks, where peers have a local only knowledge of a network in which nodes enter and leave frequently in an ad-hoc manner. For such systems, contacting large numbers of nodes is implemented by either broadcast-based schemes (e.g., [10, 11]), or *gossip*-based ap-

* 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.

proaches, e.g., [4–6]. Both produce large numbers of messages by contacting many hosts inside the network[1].

In this paper, we present the *Adaptive Group Notification* (*AGNO*) method. *AGNO* combines the utilization of state accumulated during the search process together with probabilistically stored requester addresses. Our method builds its knowledge by only monitoring the independently conducted lookups and does not require any control or membership message exchange. Finally, we show that *AGNO* achieves an efficient and adaptive performance in a variety of environments and group sizes.

2 The AGNO Protocol

2.1 Our Framework and Overview of *APS*

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 using unique object IDs. Each peer retains its own collection which is locally maintained. Ignoring physical connectivity and topology from our talk, we assume that peers are aware of their one-hop neighbors in the overlay. 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.

For our motivating example, we assume a distributed, unstructured P2P system, where peers share and request replicated resources. Objects are assumed to be updated often, like results of a live sports meeting, weather maps, stock quotes, real time aggregated statistics, 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 content holders 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, we can assume that a user frequently asking for 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, using one of the available lookup protocols for unstructured systems. The notification itself is not necessary to be directly related to the object: A severe weather alert to be effective in the next 3 hours is not related to the current area temperature object. Similarly, a notification may state that node(s) hosting security-related software will become unavailable soon (still not related to the objects shared). A change in quote prices or breaking news inside a web page are, on the other hand, directly linked to the object's

[1] Due to severe space constraints, a detailed analysis of the related work appears in the accompanying technical report [12].

content. Group communication requires a considerable amount of bandwidth, therefore content providers should wisely assess the importance of various updates/notifications and choose to push those that would be the most beneficial.

The *Adaptive Probabilistic Search (APS)* [13] method is a search scheme for unstructured overlays. Each node keeps a local index consisting of one entry for each of its neighbors. These index values reflect the relative probability of a node's neighbor to be chosen as the next hop in a future request for a specific object. Searching is based on the simultaneous deployment of k walkers and probabilistic forwarding. Indices are updated along the walkers' paths according to object discovery or failure. *APS* has proved to be an adaptive, bandwidth-efficient scheme which also provides for robust behavior in dynamic environments [14]. All nodes participating in a lookup benefit from the process, while others inherit search knowledge by proximity. Index values are refined as more searches take place, enabling the network to build a useful soft-state.

2.2 AGNO Protocol Description

The main idea behind *AGNO* is to couple search knowledge with the information necessary to contact interested peers. Thus, the equivalent of group membership is demand, 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, but for the remainder of this paper we assume 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 their neighbors. Intuitively, after a few lookup operations have been conducted, overlay paths with high index values are the ones most frequently used to connect requesters to object holders. 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 other search mechanisms, as long as they support a similar demand incentive.

APS keeps an index value for each neighbor. For *AGNO*, each peer P also maintains the index values that P's neighbors hold relative to P. If $A \rightarrow B$ 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 during the search phase: In the first case, peer P can infer the value of $X \rightarrow P$ if it knows about the index update process used and the initial value. In the explicit approach, whenever a search is conducted and X forwards to P, it piggybacks $X \rightarrow P$. 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 used by nodes during the push phase to select which neighbors will receive the notification, but they are not the only state that our method utilizes.

During searches, 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 shortcuts called *backpointers* along the search paths which point to group members. Each peer can individually decide on the maximum number of backpointers stored.

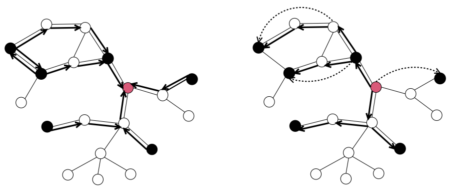


Fig. 1. Search for an object stored at the gray node and the push phase from this node towards the requesters (black nodes).

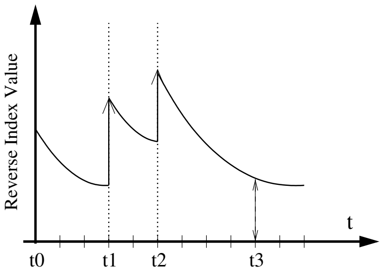


Fig. 2. Example of the reverse index value update process.

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 the notification message forwards it to its neighbors consulting the respective reverse index values. Moreover, a peer forwards directly to each of its valid backpointers with probability p_n. These messages have a $TTL = 1$ and do not travel further. Notifications are discarded either when their TTL value reaches zero or if they are received by a node more than once due to a cycle. Therefore, our scheme combines a selective, modified-BFS forwarding augmented with direct messaging (backpointers) in order to contact group members. This is shown pictorially in Figure 1.

We now discuss how the aforementioned state is maintained. 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 backpointers). 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 respective reverse index values will not be updated and will remain high.

To correct this situation, we add an *aging* factor 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. We describe this process in Figure 2. 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.

2.3 Protocol Specifics

1) Space Requirements: The amount of space required by the peers is $O(d+c)$ per object, where d is the average node degree in the overlay and c is the maximum number

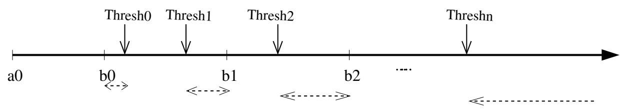


Fig. 3. Sample binning scheme with the respective threshold values for each interval.

of backpointers stored. For about 1 million objects, assuming $c = d = 4$, each peer would need approximately 48MB of memory for *AGNO*, definitely affordable by the vast majority of modern hosts.

2) Forwarding: Nodes use a threshold parameter $Thresh$ in order to choose to which neighbors the 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 similar low or high value).

3) Local Threshold Computation: Peers use a globally defined *binning* scheme to decide for the value of *Thresh*. The binning method divides the space of index values into a number of disjoint intervals. $Bin_i = ([a_i, b_i), Thresh_i)$ is characterized by its lower and upper limit values a_i, b_i ($a_0 < b_0 = a_1 < b_1 = a_2 \ldots$) and a $Thresh_i$ value. The final threshold value is $Thresh = Thresh_i$, if the average of the neighbors' reverse index values lies in $[a_i, b_i)$. Bins represent an approximation that maps reverse indices to a value representing their quality. Higher numbered bins represent higher quality indices. Figure 3 gives a graphic description of our binning scheme.

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. As a simple heuristic for selecting $Thresh_i$, their values are chosen such that $Thresh_{i-1} - b_{i-1} > Thresh_i - b_i$ and $Thresh_{i-1} < Thresh_i$, i.e., the higher the order of a bin, the smaller its threshold value is compared to the bin's upper limit.

4) Reverse Index Aging: Peers that lose interest in an object should be left out of the push phase as quickly as possible. Our scheme uses an 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 at 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. 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 a value for ξ which satisfies: $(1-\xi)^T \max_i(Thresh_i) < \min_i(Thresh_i)$ (1). In effect, we pick ξ such that any reverse index with value less or equal to $max_i(Thresh_i)$ will be reduced below the lowest threshold (and thus will not be selected) if not used for T time steps. In the vast majority of cases, notifications are considerably less frequent than requests, therefore we set $T = O(1/\lambda_r)$. This is done in order to quickly identify and decrease idle indices in the overlay. The maximum $Thresh_i$ represents the smallest high-quality index value in our binning scheme. Therefore, 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 threshold values or smaller T values results in a more aggressive aging.

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 (F_0 and F_1 respectively) 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., [15, 16]). We use one of the schemes in [15], which requires only $O(1/\varepsilon^2 + \log(m))$ memory bits, 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: 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. Given that a peer's threshold bin is i at time t, the probability with which each stored backpointer will be followed is p_{n_i}, given from the set $\{p_{n0}, p_{n1}, ... p_{n_i}, ...\}$ (i.e., one p_n value for each bin). We choose those values such that $p_{n_i} > p_{n_{i+1}}$, 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.

3 Simulation Results

3.1 Simulation Methodology and Compared Methods

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, utilizing *BRITE* [17] and *Inet-3.0* [18] to create these overlays respectively.

The following metrics are used 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: Given the value of 5 as an estimate for the TTL parameter[19], we set $p_r \geq \frac{1}{TTL}$. Given the index update policy used by *APS* as described in [13], we employ a simple 3-bin scheme. The first bin represents indices below the initial value

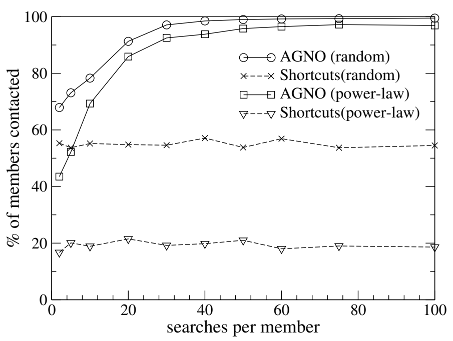


Fig. 4. Success over variable number of searches.

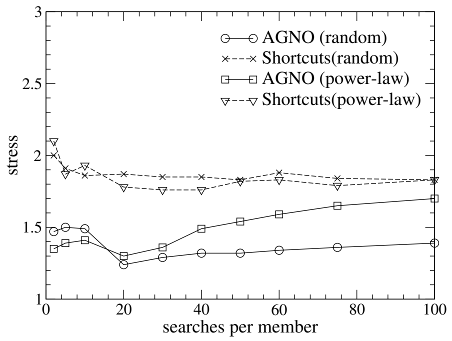


Fig. 5. Stress over variable number of searches.

(no successes), the second those with some hits and the last those with more successes. By default, we set $c \simeq d$ (which reserves an amount of space for backpointers roughly equal to the average node degree) and $T = 2T_r$. Thus from (1) 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 $\varepsilon = 0.1$. A more detailed description of the parameters chosen as well as experimental evaluations with different parameter/bin selections can be found in [12].

Compared Methods: We compare our method against 3 algorithms: The SCAMP membership protocol [5] and the two rumor-spreading schemes in [6]: *Rumor Mongering* (RM) and its deterministic version (det-RM), where peers have complete topology information. In SCAMP, 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 [6], 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.

Finally, we also implement a pure shortcut forwarding scheme (*Shortcuts*), where backpointers are stored as in *AGNO*, while in the push phase a peer forwards to all valid shortcuts, using the standard *TTL* scheme. 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.

Table 1. (Success, Stress) results for the remaining methods using 500 requesters.

	SCAMP	**RM**	**det-RM**
10K Random	(89%, 2.7)	(89%, 34.5)	(98%, 31.1)
10K Power-law	(68%, 2.1)	(27%, 13.6)	(65%, 10.8)

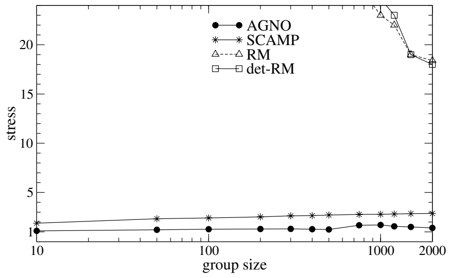


Fig. 6. Stress over variable group size.

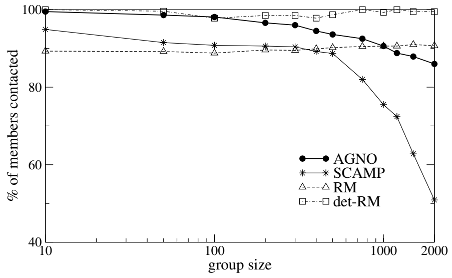


Fig. 7. Success over variable group size.

3.2 Basic Performance Analysis

In this first set of experiments, using a group of 500 requesters, we vary the number of lookups each of them makes before a single push phase occurs. We report the results averaged over 10,000-Node random and power-law topologies. Figures 4 and 5 present the results for *AGNO* and *Shortcuts* which are affected by the number of searches.

We notice that the pure shortcut scheme cannot provide an efficient notification method by itself. *AGNO* quickly contacts the majority of requesters after only a few searches take place, while maintaining a low stress factor. As our scheme creates better 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.

In the power-law topologies, where about 34% of the peers have degree one, fewer paths are used compared to the random graphs. This, combined to the fact that $\xi = 0$ in these experiments, explains why the stress for *AGNO* slightly increases with more requests. The respective results for the remaining methods (not affected by searches) are shown in Table 1. *AGNO* proves very accurate (in the big majority of runs) and also the most bandwidth-efficient 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 [6].

Next, we measure the scalability of our method with 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. For SCAMP, the membership protocol is run before each different group size. Figures 6 and 7 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

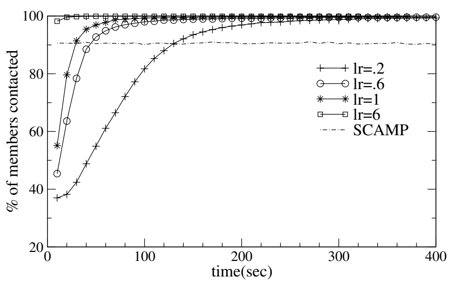


Fig. 8. Success over variable λ_r values ($T_n = 10sec$).

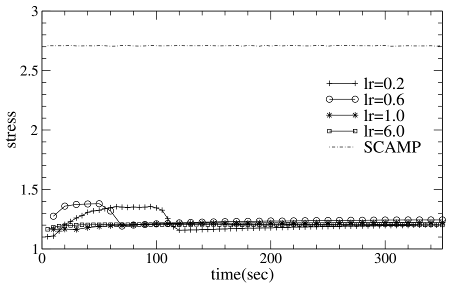


Fig. 9. Stress values over variable λ_r values ($T_n = 10sec$).

for some requesters. The RM schemes produce a similar number of messages regardless of the group size, 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.

3.3 Sensitivity to λ_r

Assuming a group size of 1,000 peers, we try to evaluate the performance of *AGNO* for different λ_r values. Figures 8 and 9 show the results. Not surprisingly, the larger 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 stable 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.

3.4 Changes in Group Sizes

Figure 10 shows how our two metrics are affected by having 10%–80% of the 1,000 requesters leave the group (stop making queries) at time $t = 100sec$. We assume (worst-case scenario) that all these nodes jointly and instantly decide to leave 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. On the other hand, the success rate is hardly affected.

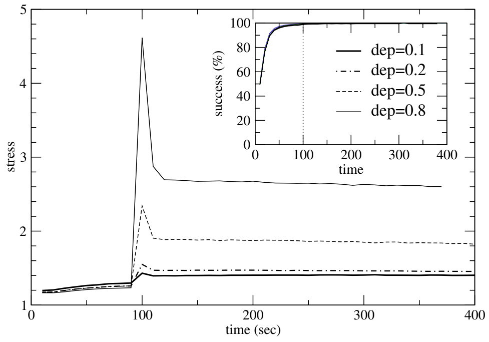
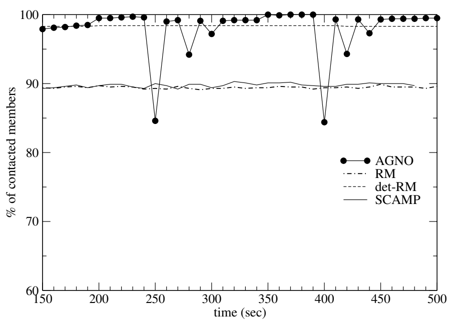


Fig. 10. Stress and success rates when a different ratio of peers depart at time t=100sec ($\lambda_r = 1, T_n = 10sec$).



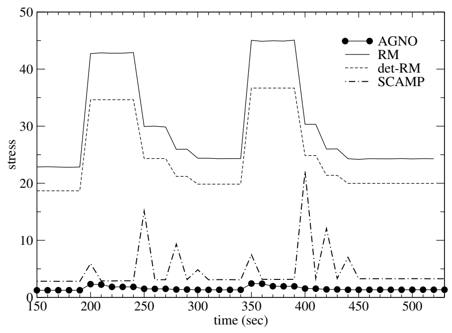


Fig. 11. Success after a series of member departures and arrivals ($\lambda_r = 0.5, T_n = 10$).

Fig. 12. Stress after a series of member departures and arrivals ($\lambda_r = 0.5, T_n = 10$).

Figures 11 and 12 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 $\lambda_r = 0.5/sec$, 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 presence 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).

4 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 showed that *AGNO* adapts quickly to variable request rates and group sizes, being at least twice as bandwidth-efficient as the compared methods.

References

1. Chu, Y., Rao, S., Seshan, S., Zhang, H.: Enabling conferencing applications on the internet using an overlay muilticast architecture. In: SIGCOMM. (2001)
2. Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable application layer multicast. In: SIGCOMM. (2002)
3. Zhuang, S., Zhao, B., Joseph, A., Katz, R., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: NOSSDAV. (2001)
4. Datta, A., Hauswirth, M., Aberer, K.: Updates in highly unreliable, replicated peer-to-peer systems. In: ICDCS. (2003)
5. Ganesh, A., Kermarrec, A., Massoulie, L.: SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In: Networked Group Communication. (2001)
6. Portmann, M., Seneviratne, A.: Cost-effective broadcast for fully decentralized peer-to-peer networks. Computer Communications **26** (2003)
7. Rowstron, A., Kermarrec, A., Castro, M., Druschel, P.: Scribe: The design of a large-scale event notification infrastructure. In: NGC. (2001)
8. Jannotti, J., Gifford, D., Johnson, K., Kaashoek, F., O'Toole, J.: Overcast: Reliable multicasting with an overlay network. In: OSDI. (2000)
9. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable networks. Lecture Notes in Computer Science (2001)
10. Gnutella website: http://www.gnutella.com.
11. Kalogeraki, V., Gunopulos, D., Zeinalipour-Yazti, D.: A Local Search Mechanism for Peer-to-Peer Networks. In: CIKM. (2002)
12. Tsoumakos, D., Roussopoulos, N.: AGNO: An Adaptive Group Communication Scheme for Unstructured P2P Networks. Technical Report CS-TR-4590, University of Maryland (2004) (ext. version) http://www.cs.umd.edu/ dtsouma/objects/pbroad.pdf.
13. Tsoumakos, D., Roussopoulos, N.: Adaptive Probabilistic Search for Peer-to-Peer Networks. In: IEEE Intl Conf. on P2P Computing. (2003)
14. D.Tsoumakos, Roussopoulos, N.: A Comparison of Peer-to-Peer Search Methods. In: WebDB. (2003)
15. Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D., Trevisan, L.: Counting distinct elements in a data stream. In: RANDOM. (2002)
16. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC. (1996)
17. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: An Approach to Universal Topology Generation. In: MASCOTS. (2001)
18. Jin, C., Chen, Q., Jamin, S.: Inet: Internet Topology Generator. Technical Report CSE-TR443-00, Department of EECS, University of Michigan (2000)
19. Ripeanu, M., Foster, I.: Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In: IPTPS. (2002)