Adaptively Routing P2P Queries Using Association Analysis

Brian D. Connelly, Christopher W. Bowron, Li Xiao, Pang-Ning Tan, and Chen Wang

Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824 US

{*connel42,bowronch,lxiao,ptan,wang*}@*cse.msu.edu*

Abstract

Unstructured peer-to-peer networks have become a very popular method for content distribution in the past few years. By not enforcing strict rules on the network's topology or content location, such networks can be created quickly and easily. Unfortunately, because of the unstructured nature of these networks, in order to find content, query messages are flooded to nodes in the network, which results in a large amount of traffic. This work borrows the technique of association analysis from the data mining community and extends it to intelligently forward queries through the network. Because only a small subset of a node's neighbors are forwarded queries, the number of times those queries are propagated is also reduced, which results in considerably less network traffic. These savings enable the networks to scale to much larger sizes, which allows for more content to be shared and more redundancy to be added to the system, as well as allowing more users to take advantage of such networks.

I. Introduction

The popularity and number of peer-to-peer (P2P) networks has exploded in the past several years. They have proved to be a viable method for the dissemination of data across a network. Aside from the legal issues faced by a few existing networks regarding the distribution of copyrighted material, P2P networks also serve many useful legitimate purposes, such as load balancing, providing more flexible and up-to-date routing information [1], managing voice traffic [2], and offering efficient downloads of free software [3].

Many of the networks in use today follow the model of unstructured peer-to-peer, which was first widely used in the Gnutella [4] network. These networks do not impose any rules as to how the nodes organize themselves or where shared content is located. This has the benefit of allowing nodes to join and leave the system without significantly affecting the entire system.

One disadvantage of this approach, however, is that the location of content shared on the network is not known. In order for a user to find a particular piece of content, he or she ”floods” the network with query messages. In flooding, a query message is sent to all of a peer's neighbors, which, in turn, forward the query to all of their neighbors, and so on. This behavior results in the query reaching all nodes, so if any node shares content that matches the user's query, it will be found. Because flooding creates so many messages, the amount of traffic on the network grows considerably with each node that joins, because that node will propagate all received queries to each of its neighbors, as well as issue new queries, which generate many flooded query messages. The end result of this large volume of traffic is that current networks using unstructured P2P reach a limit in the number of users who can concurrently use the system.

This paper presents a new approach to limiting the number of queries which are flooded in the network. This approach uses the concept of association analysis, which has been studied extensively in the data mining community. By extending association analysis to include measures of quality for rule sets and driving the rule generation process by feedback, nodes intelligently forward query messages to a subset of neighbors that are likely to continue forwarding queries towards nodes that share the desired content. Because this significantly reduces the number of query messages that are flooded while maintaining the ability to successfully locate content, the overall traffic on the network is decreased, allowing more users to make use of the network simultaneously. As this approach does not change the protocol used by unstructured P2P networks, it can be deployed in nodes in current systems without requiring that all nodes support this method.

The structure of this paper is as follows: Section II selects and discusses some of the work that has been done to lessen the effects of flooding. Section III introduces the concept of association analysis and some of the measures used to determine the quality of the results that are generated. Additionally, a family of algorithms which take advantage of association analysis is described. The design and simulation methodologies used to validate these algorithms are detailed in Section IV, and the results from simulations are presented in Section V. Finally, Section VI draws conclusions from the work that has been done and introduces future directions that could be investigated.

II. Related Work

Much work has been done to address the problem of flooding in unstructured peer-to-peer networks. These works usually fall into one of three categories: limiting the scope of a query so that queries are flooded less frequently, forwarding the query to a select subset of neighbors instead of flooding, and re-designing the network so that flooding doesn't have to be performed.

Several ideas have emerged in the first category that have benefited the system. The first is associating a time-to-live (TTL) value with each query, which sets the number of times that a query can be forwarded. A second technique is expanding-ring search [5], which gives query messages a low TTL and increases the TTL if the queries continue to not find content. Although these techniques can decrease the number of query messages sent through the network, limiting the scope of a query can prevent it from finding a host. Also, because expanding ring searches increase TTL until a hit is found, nearby nodes may receive the query several times, which is an increase in traffic.

Because of the shortcomings of approaches that limit the scope of queries, new methods have emerged which fall within the second category. By forwarding queries to a select subset of a node's neighbors, the amount of traffic is reduced not only at those nodes, but down the line, as less propagation is done. k-random walks [6] select one or more neighbors to which messages will be sent. This approach allows the TTL for queries to be quite large, since the number of messages required to support this is lower. As paths are found to be unsuccessful, a node may choose new neighbors to use. This approach may require more time to locate the content, as the number of nodes being searched at a given time may be much smaller.

Another set of approaches that fits within this second category exploits the concept of interest-based locality, which states that because users have a limited set of interests, a node that has provided hits previously is likely to share the same interests, and since the user is likely to query for content that falls within these interest groups, the overlap may mean that the other node shares this new content. By keeping a list of shortcuts to other nodes [7], a peer first sees if any of its shortcuts share the content that is being looked for before resorting to flooding. More advanced uses of this idea [8] [9] build additional, higher-level topologies, allowing nodes with similar interests to be ”close” to each other.

Finally, a different approach within this category forwards query messages to a small subset of neighbors based on the estimated ”goodness” of those neighbors [10], which is similar to the basis for the work presented in this paper. By keeping a table of each neighbor node and the number of documents classified within a defined set of topics that are reachable via that neighbor, a node forwards a query on to the neighbor estimated to lead to the most number of documents whose topics match those in the query.

Considerable work has been done in the third category as well, and the most notable contributions to come out of this are the structured category of P2P networks such as CAN [11], Chord [12], and Pastry [13]. These networks impose a set of rules which govern where the nodes may be located in the topology, as well as where the files or indices are stored. Queries can efficiently find content by following the rules of the system. However, the rigid structure of the network complicates node joins and departures, and if a certain set of the nodes fail simultaneously, the network can become disconnected. Another problem is that queries must match the content exactly, so wild card searches or searches which contain a permutation of the words will not find the corresponding content.

In an additional class of networks that imposes structure upon the network, nodes connect to a ”superpeer” that maintains an index of the contents of each node connected to it [14]. When a node issues a query, it first sends the message to its superpeer, which compares the query to its index. If one of the other nodes attached to that superpeer is sharing the desired content, the querying node is notified of this and can initiate a download from the hosting peer. If none of the nodes connected to that superpeer hosts content matching the query, the superpeer then floods the query to the other superpeers, which compare the query with their indices. Although this approach has the benefit of reducing the number of hops required for queries, it can still suffer from the effects of flooding on larger systems.

III.Routing Queries with Association Analysis

This section describes how association analysis can be used to make routing decisions. We briefly introduce association analysis and then discuss the theory behind its use in query routing. Finally, a family of algorithms are presented that generate and use rules to route incoming query messages.

A.Association Analysis

Association analysis is the process of extracting interesting relationships hidden in large data sets [15] [16]. Since its introduction, considerable research has been performed within the data mining community to expand this work and implement it for use in various problem domains.

The relationships extracted by association analysis, referred to as *association rules*, are represented in the form $\{A\} \rightarrow \{B\}$, and can be interpreted as: if event A occurs, B is likely to also occur. The event on the left side of the rule is known as the *antecedent*, and the event on the right is known as the *consequent*. Both antecedents and consequents can contain more than one event, so association rules of the form $\{A, C, D, F, H\} \rightarrow \{B, N\}$ may also be mined.

One classical application for association rule mining is market basket analysis. For example, the transactions made at a grocery store are analyzed, and it is discovered that people who buy diapers also tend to buy beer. A rule for this phenomenon would be written as $\{Diapers\} \rightarrow \{Beer\}$. Such rules could be used by managers in determining how to stock their shelves. Perhaps by placing beer in an aisle near diapers, sales of beer might increase.

When using association analysis, many rules may be generated, and it is important to evaluate their usefulness. As another example, it may be found that people who buy caviar also tend to buy sugar ($\{Caviar\} \rightarrow \{Sugar\}$). Although this rule is interesting, it is not particularly useful, because people rarely buy caviar. In order to quantify this usefulness, the measures of *support* and *confidence* are used [15].

Support is the fraction of all transactions that contain both the antecedent and consequent. The support is likely to be low for the rule $\{Caviar\} \rightarrow \{Sugar\}$, because purchases containing both of these items are quite rare. However, purchases containing diapers and beer occur much more often, so the support for $\{Diapers\} \rightarrow \{Beer\}$ will be higher.

Confidence measures how often the consequent occurs in transactions containing the antecedent over all transactions containing the antecedent. If a high percentage of the transactions containing caviar also contain sugar, then the confidence of the $\{Caviar\} \rightarrow \{Sugar\}$ rule will be high.

Both measures provide valuable information about the quality of the rules, so each must be considered. Pruning, the process of removing association rules, may be done by removing any rules where either measure falls below a threshold. The choice of thresholds will depend on the application domain and the data available.

B.Design Rationale

This paper borrows the concept of association analysis and applies it to the problem of flooding in decentralized, unstructured peer-to-peer networks. The basic idea of this approach is for each node to generate a rule set based on query and reply messages that it has received in the past and use these rules to forward future queries. For example, if the node receives a query message from one of its neighbors, forwards it on, and later receives a reply message for that query, it now knows that because it forwarded the query to a particular neighbor, that path led to a node on the network which had content matching the query.

Interest-based locality tells us that a node that has satisfied a query for a particular piece of content will likely be able to satisfy future queries for content that fall within the same interests as the first. Because of this, future queries should try to reach the same server node to take advantage of interest-based locality. If the query follows the same path, it will arrive at that node quickly and without having to travel along many unnecessary paths. This can be accomplished by simply routing the query to the next hop on that path, instead of sending it along all paths.

A secondary benefit of this approach is that all nodes in the network do not need to support this routing method in order for one node to use it, although the benefits increase as the number of nodes using this routing technique increases. Additionally, this technique preserves the anonymity of the host issuing the query, which is another characteristic of many unstructured P2P networks. Finally, if hits aren't found for a particular query when using this approach, the node can still revert to flooding, so the quality of the search results should not decrease dramatically.

1) Association Rules for Query Routing: The association rules that will be generated for use in query routing are of the form $\{host1\} \rightarrow \{host2\}$, where $host1$ is a neighbor from which the node receives queries, and $host2$ is a neighbor who is the next hop on a path that has led to hits for earlier queries coming from $host1$. Because both the antecedent and consequent contain only one item, rule generation and rule set pruning can be done easily.

First, hosts forwarding a query are paired with the hosts that respond to the queries. From this list of hosts, pairs are removed that have been used less than a set number of times, a process which is known as *support pruning*. This number can be set depending on the quality of the desired rule sets. If this threshold is set low, many rule sets may be generated and used, as more pairs will be included. If the threshold is set high, the number of rule sets generated may be much lower. Although this would seem to result in smaller, higher-quality rule sets which yield comparable results to larger rule sets, this may not necessarily be the case, because future queries which use this concise rule set may come from hosts not in the rule set, and users may query for content from other interest areas.

It is often the case that more than one rule containing a given antecedent will exist, meaning that forwarding a query received from that node to more than one neighbor node resulted in previous hits. In these situations, future queries can either be sent to a random subset of neighbors as with k-random walks, or sent to the k neighbors with the highest support.

2) Measuring the Quality of Rule Sets: The quality of the resulting rule sets must be properly measured in order to determine the effectiveness of this approach. Unfortunately, the traditional measures of support and confidence are not entirely descriptive for this application. This is due to the fact that they are measures for individual rules in the rule set, not the rule sets as a whole. Using these measures would focus on the hosts from which queries come, instead of the success of query routing based on the the rule sets that have been generated.

We define two new measures to address the number of queries related to generated rule sets. The first measure, *coverage*, is denoted by α, and is described in Equation 1.

$$\alpha = \frac{n}{N} \tag{1}$$

In Equation 1, N represents the total number of unique queries for which there is a response received within the test set, and n denotes the number of unique queries for which there exists an antecedent in a rule that matches the source of that query.

The second measure, *success*, is denoted by ρ, and is calculated using Equation 2.

$$\rho = \frac{s}{n} \tag{2}$$

In this equation, s represents the number of queries for which the host issuing the query and the neighbor sending a reply message are antecedent and consequent, respectively, of a rule in the rule set. This can be thought of as the fraction of all rules matching the host issuing the query that also contain a node that would result in a hit, should the query be forwarded to that node, following the rule.

The range of values for α and ρ are between 0 and 1, inclusive. These values can also be thought of as percentages, so a coverage value of 0.9 indicates that rules exist for 90% of queries issued.

If the success is low, this means that the rules matching the node that forwarded a query would not forward that query on towards a node that would result in a hit. Alternately, if the success is high, this means that by following the rules in the rule set, the number of queries that would be sent towards a node which contain content matching the query is high.

Of course, success could be high, and the rule sets generated could still be of little help to the system. This would be the case when success is high, yet coverage is low. In other words, the queries matching rules would be forwarded correctly, but since the number of queries matching rules is low, the number of queries that could be resolved using this method is also low. On the other hand, if coverage is high, but success is low, this would mean that the rule set has rules for routing many of the incoming query messages, but the rules would be forwarded to the wrong neighbors, resulting in query misses. Because of situations like these, both coverage and success must be high for a rule set to be effective when using this method.

3) The Static Ruleset Approach: Static Ruleset is the most simple way in which association rules can be used to forward queries. In this approach, a rule set is created by combining query and reply messages seen within a fixed amount of time, and less-frequently-used pairs are pruned. This rule set is then used for all subsequent query messages.

The following pseudocode describes the operation of Static Ruleset:

```
STATIC-RULESET
R ← GENERATE-RULESET
for each block b
    do RULESET-TEST($R, b$)
```

The benefit of Static Ruleset is its simplicity, and its main shortcoming is its lack of flexibility. If the types of content queried for or the neighbors issuing the queries change over time, the rules may not accurately match these new situations. As peer-to-peer networks are usually highly dynamic, this is likely to quickly be the case.

4) The Sliding Window Approach: The second approach takes the dynamic nature of the network and queries issued into consideration. Instead of generating one rule set and using it for all subsequent queries, this method generates a rule set, uses it for a fixed number of queries, and then generates a new rule set based on the previous query and reply messages.

The periodic re-creation of the rule set allows Sliding Window to maintain up-to-date information about the nodes issuing queries and the nodes which reply to them. Should the topology of the network change, Sliding Window would adjust to this in the next block of messages.

Pseudocode for the operation of Sliding Window is listed below:

```
SLIDING-WINDOW
for each block b
    do R ← GENERATE-RULESET($b - 1$)
       RULESET-TEST($R, b$)
```

The disadvantage of Sliding Window is that it may update the rule set too frequently. Although rule set generation is a fairly simple procedure, it may not be necessary if the rules used still apply to the queries being issued.

5) The Lazy Sliding Window Approach: The Lazy Sliding Window approach attempts to maintain an up-to-date rule set like Sliding Window, but without generating new rule sets as frequently. Instead of updating the rule set after every block, this approach updates after the rule set has been used for a fixed number of blocks. The frequency of updates is a parameter which can be changed depending on how closely the rule set should follow changes in the network.

The following pseudocode implements the Lazy Sliding Window algorithm, generating new rule sets every 10 blocks:

```
LAZY-SLIDING-WINDOW
$trial\_num \leftarrow 0$
for each block b
    do R ← GENERATE-RULESET($b - 1$)
       $trial\_num \leftarrow trial\_num + 1$
       RULESET-TEST($R, b$)
       if $trial\_num$ $mod$ $10 = 0$
          then R ← GENERATE-RULESET($b$)
```

The benefit of Lazy Sliding Window is the reduction in the number of rule sets that have to be generated. However, since rule sets are generated less frequently, they will not track changes as closely as with Sliding Window. This could cause the coverage and success for Lazy Sliding Window to be less than those of Sliding Window, which results in the number of queries that must be resolved using flooding to increase. Depending on the system, a small increase may be acceptable; however, in larger systems, we still wish to reduce the number of flooded queries as much as possible.

6) The Adaptive Sliding Window Approach: The final approach, Adaptive Sliding Window, attempts to maintain the high coverage and success values possible with Sliding Window with the reduction in the frequency of updates seen in Lazy Sliding Window. To accomplish this, Adaptive Sliding Window adds thresholds for the coverage and success of a rule set. In order to capture the dynamic nature of the network, these thresholds are constantly updated so that threshold values remain reasonable for all states of the network. One simple method would be to use the mean of the previous N values. If either the coverage or success obtained when using the rule set fall below the threshold, a new rule set will be generated. This has the effect of maintaining high-quality rule sets and updating the rule set only when necessary.

The following pseudocode shows the operation of Adaptive Sliding Window:

```
ADAPTIVE-SLIDING-WINDOW
for each block b
    do R ← GENERATE-RULESET($b - 1$)
       $ct$ ← CALC-COVERAGE-THRESHOLD($b - 1$)
       $st$ ← CALC-SUPPORT-THRESHOLD($b - 1$)
       $results \leftarrow$ RULESET-TEST($R, b$)
       if $results[coverage] < ct$
          then R ← GENERATE-RULESET($b$)
          else if $results[success] < st$
                  then R ← GENERATE-RULESET($b$)
```

IV. Methodology

The concept of routing queries based on association rule sets was thoroughly tested to determine how well the different methods worked in real-life situations. Also of interest was how the values for different parameters used by those methods affected the quality of the rules generated. First, the data used for these experiments are introduced in Section IV-A. The simulation program that was used to test these methods is described in Section IV-B.

A. P2P Trace Data

Trace data were collected for queries and replies sent to a modified node in the Gnutella network over a 7-day period. For queries, the query string, the time of the query, the IP address of the node that forwarded the query, and a globally-unique identifier (GUID) assigned to the query by the issuing node were recorded. For replies, the time the reply was received, the GUID of the query, the neighbor from which the reply was sent, the host of the matching file, and the name of the file matching the query were recorded.

During the process of importing the data into a relational database, it was discovered that some of the globally-unique identifiers were not truly unique, and instances of different queries having the same GUID were found. Clients that did not properly generate GUIDs are assumed to be the cause of this. For these instances, only the record corresponding to the first use of that GUID was kept. After removing those containing duplicate GUIDs, a total of 10,514,090 query messages and 3,254,274 reply messages were deposited into the database.

A table was created to house pairs of query messages received by the node that collected the data and the reply messages received in response to those queries. The join of these data produced 3,254,274 query-reply pairs.

Several additional tables were defined to store temporary results from the simulator. These tables stored data such as the current rule set and the block of data being used to test the rule set. Roughly 2.6 gigabytes of disk space was required by the fully-populated database.

B. The Query Simulator

A simulator was written to generate rule sets based on the data in the database and test the generated rule sets against different query-reply pairs. This simulator was written in under 500 lines of code using PHP [17], which was chosen for its simplicity and powerful interface to databases.

Based on the parameters given, the simulator builds a rule set based on a defined number of query-reply pairs (”blocks”) and tests those rules against one or more proceeding blocks, depending on the method chosen for the simulation.

The database table representing the rule sets contains three values for each entry: the host from which one or more queries were received, a node that returned a reply message in response to one of those queries, and the number of times that that node sent reply messages in response to queries sent from the node that forwarded the query. The quality of the rules created can be controlled by a parameter whose value represents the minimum number of times the source of the query message and the host that returned a reply message are used within a given block. Any pairs of hosts that have not been seen that many times are removed from the rule set.

The time required for simulations to complete varied depending on the method used in the simulations and the values of the parameters used. After creating indices to frequently-searched fields in the database, most simulations required roughly 45 minutes to complete when run with block sizes of 10,000 query-reply pairs. As the block size was increased, the amount of time required also increased, mostly in part to the larger number of join operations performed. One simulation using Static Window with a block size of 50,000 took approximately 3 hours to complete.

V. Simulation Results

A total of 22 simulations were run in order to test the accuracy of the methods developed, as well as to demonstrate how changing the parameters related to those methods affects the results. The following sub-sections describe how each of the proposed algorithms performed. Unless otherwise noted, the threshold for pruning query-reply pairs was 10, and the block sizes were 10,000 query-reply pairs. On average, rule set generation required no more than a few seconds.

A. Static Ruleset Results

As was predicted, using Static Ruleset resulted in rule sets being generated that had high coverage and success for the first few blocks of data that they were tested against. However, both of these measures dropped quickly. In fact, once the success had dropped to almost 0 around the 16th trial, it never rose again. Coverage, on the other hand, dropped, but remained around 0.4 for several more trials. This indicates that some of the same hosts were still forwarding queries, but the hosts that the rule set would suggest forwarding to would not lead to hits. Over the 365 trials performed, the average coverage was 0.18, and the success was under 0.02, which confirms that Static Ruleset does not perform well over time due to the dynamic properties of the network. Additional simulations performed with varying block sizes yielded very similar results.

B. Sliding Window Results

Simulations showed that Sliding Window performs very well. For the simulation results shown in Figure 1, the average coverage was over 0.80, and the average success was just under 0.79, demonstrating that Sliding Window can result in a large reduction in the number of query messages that need to be flooded.

Because Sliding Window has two parameters—the size of the block used and the minimum threshold used when pruning query-reply pairs—additional simulations were run to determine how changes in these values affect the coverage and success of Sliding Window. Adjustments to the block size affect the reactiveness of the algorithm. Although Sliding Window considers more hosts when generating rulesets with larger blocks, a longer amount of time has elapsed, meaning some rules may be stale. Smaller blocks result in rules which contain more recently-seen hosts, but may have less support. The coverage of Sliding Window when used with different block sizes is shown in Figure 2. The results of other simulations are not shown in this paper. The data do show that Sliding

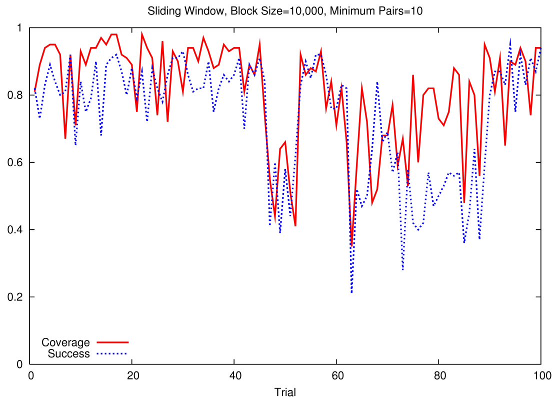

Fig. 1. Coverage and Success of Sliding Window over time

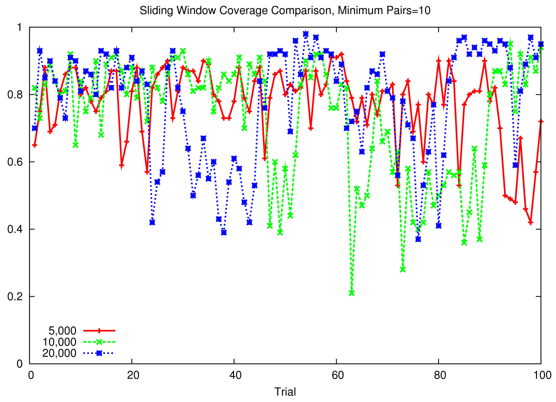

Fig. 2. Coverage Sliding Window over time using different block sizes.

Window achieves very similar levels of coverage when either the block size or the query-reply pair threshold is altered. This demonstrates that only a small number of query-reply pairs are needed to successfully forward the majority queries without flooding.

C. Lazy Sliding Window Results

Lazy Sliding Window performed as expected. Following rule set generations, coverage and success values were high, and they tapered down as time passed. A typical set of simulation results are shown in Figure 3. For this simulation, the average coverage and success values were each 0.59, which is considerably greater than those of Static Ruleset, and less than those of Sliding Window. Further simulations were performed with varying block sizes, and similar results were found for each.

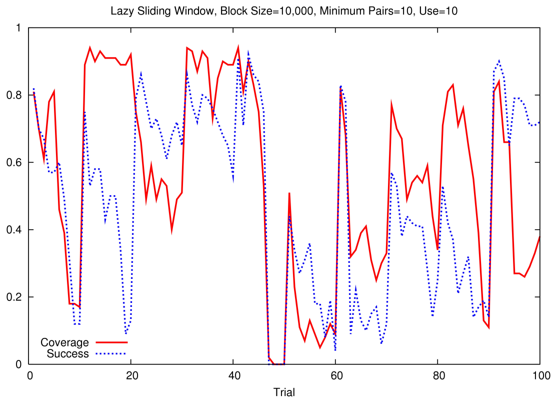

Fig. 3. Coverage and Success of Lazy Sliding Window over time. Each generated rule set was used for 10 blocks.

D. Adaptive Sliding Window Results

Simulations using Adaptive Sliding Window performed very well. Following rule set generations, coverage and success values were high, and a slight drop in their values could be seen as the rule sets were used for subsequent blocks. Because of the thresholds, however, the decreases in coverage and success were never dramatic.

Figure 4 shows the coverage and success values obtained from a typical simulation. The average coverage was 0.78, and the average success was 0.77. used with a threshold of 0.7 for both coverage and success. On average, new rule sets were generated every 1.7 blocks.

When the number of previous values used for threshold calculation was increased to 50, rule sets were generated every 1.9 blocks, which is almost half as many rule set generations as Sliding Window. The average coverage value for this simulation was 0.79, and the average success was 0.76, which comes very close to values obtained in Sliding Window experiments.

VI. Conclusions and Future Work

Simulations of the methods introduced in this paper show that selectively forwarding query messages based on rules generated through the use of association analysis can lead to a dramatic reduction in the number of queries that are flooded. Because of this, results to queries may be received more quickly, and the networks can support more simultaneous queries, allowing the number of users who can efficiently and successfully use the network to grow.

The creation of rule sets from streams has also been investigated in the data mining community [18]. An additional algorithm is currently in development that would

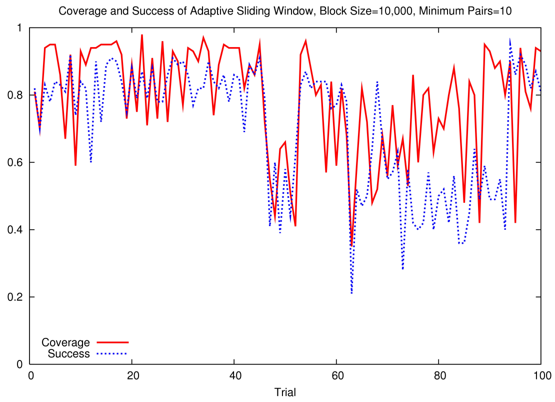

Fig. 4. Coverage and Success of Adaptive Sliding Window over time using the previous 10 values for threshold calculation.

create rule sets for query routing and update these rules immediately as query and reply messages are received. This method will allow rules to constantly be kept up-to-date without the overhead created by periodically generating entire rule sets. Initial simulations have been very promising, and consistently show coverage and success values above 90%.

The addition of confidence-based pruning to the rule generation stage could be one way of reducing the size of rule sets while retaining high coverage and success. Adding dimensions such as the query strings during rule generation and then clustering based on this information could also aid in increasing the quality of the rule sets.

One dramatic use for the rules generated using association analysis would be to re-arrange the topology of the overlay network. For example, instead of forwarding query messages to a neighbor, which will in turn forward the message on to one of its neighbors, a node could ask its neighbors to which node they would forward queries from it. Once the node has this information, it could attempt to make this third node a new neighbor, which would result in queries being forwarded in the future requiring one less hop in the path to its target.

Methods introduced in this paper could be successfully used in conjunction with other approaches. For interest-based shortcuts, association rules could be used to route queries that have not been successfully replied to when using the shortcuts. This would serve as one last chance to avoid flooding. Another way in which these two approaches could be combined would be to use association rules to manage which shortcuts the nodes maintain. Although that solution would not take advantage the methods shown in this paper, it is a potential application for association rules which may lead to higher success rates when using shortcuts. Additionally, association rules could be kept for each interest group when used with association overlays. This would allow for efficient querying within each interest group, as well as possibly improving inter-interest-group queries if a rule set for this purpose were also maintained.

References

[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Banff, Canada, 2001.

[2] Skype, "http://www.skype.com."

[3] B. Cohen, "Bittorrent, http://www.bittorrent.com."

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

[5] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th international conference on Supercomputing, New York, New York, USA*, 2002, pp. 84–95.

[6] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proceedings of IEEE INFOCOM*, 2004.

[7] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *Proceedings of IEEE INFOCOM*, 2003.

[8] E. Cohen, A. Fiat, and H. Kaplan, "A case for associative peer to peer overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 95–100, 2003.

[9] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "Fast and low-cost search schemes by exploiting localities in P2P networks," *Journal of Parallel and Distributed Computing*, vol. 65, no. 6, pp. 729–742, 2005.

[10] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of the 28th Conference on Distributed Computing Systems*, 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proceedings of ACM SIGCOMM*, 2001.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2001, pp. 149–160.

[13] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, p. 329, 2001.

[14] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *Proceeedings of the IEEE International Conference on Data Engineering*, 2003.

[15] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, P. Buneman and S. Jajodia, Eds., Washington, D.C., 26–28 1993, pp. 207–216.

[16] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," in *Special Issue on Learning and Discovery in Knowledge-Based Databases*, N. Cercone and M. Tsuchiya, Eds. Washington, U.S.A.: Institute of Electrical and Electronics Engineers, 1993, no. 5(6), pp. 914–925.

[17] R. Lerdorf, "PHP: Hypertext preprocessor, http://www.php.net."

[18] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS)*, 2002.