Fast Resource Information Dissemination in a P2P Network

Ashish Gupta, Ankit Mohan, Ananth Sundararaj
Northwestern University
Date: 3/20/2003

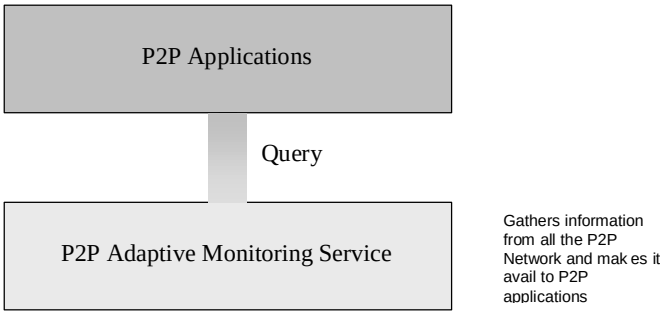
Under guidance of
Prof. Fabian Bustamante
Course: Advanced Operating Systems

Introduction

We look at the design of a peer-to-peer resource monitoring system, in which every peer in the network may be interested in knowing some resource characteristics about the rest of the peers. Potentially, availability of such information will provide P2P applications with better knowledge about the P2P network and can exploit this information for their tasks**.** The goal of this project is to make the various characteristics of the P2P system readily available to P2P applications by building an efficient P2P Monitoring Service, helping them to exploit the P2P system in an intelligent way.

Layered Model

Our aim is to provide a generic service, which is not application specific. The resource monitoring service (RMS) can be running on all hosts and applications may query it to request useful information for its purpose. This corresponds to a layered model as shown below:



Applications

Some applications, which can be conceived over such a service, are:

1. Peer-to-Peer Load Balancing
Embarrassingly parallel applications may be distributed by a peer among its fellow peers efficiently using resource information. E.g. Peer-to-Peer Ray Tracing application

2. P2P Computing or Network Shared Computing
This is a new application, which we thought could be built over a resource-monitoring infrastructure. Conventionally, a master host is responsible for carrying out a computation task and it may further distribute the task to others. Our idea is to develop a network based computing model, where no single computer is responsible but the entire network is responsible for the computation. Using readily available resource information a peer can effectively partition its tasks and distribute them.

Efficiency

In our resource monitoring system, there are two points to note:

1. Every peer needs to send information to everybody else! This separates it from other information dissemination methods like single-source multicast. Therefore a major efficiency concern is that it needs to be scalable in terms of network resource usage.
2. Since we are disseminating resource information, the “freshness” of this information available to peers is also a concern. Resource information (depending on the application) can get old with time, and hence an efficient resource monitoring system needs to provide some guarantees on the “freshness” of information for it to be useful.

These two properties are captured by some metrics which be define later.

How to tackle both is an interesting problem. There seem to exist tradeoffs between these two properties. If we have a scalable system in terms of network resource usage, its guarantees about freshness may be quite poor and vice versa.

Communication Model

We consider gossip communication as the communication model for resource information dissemination. Since the problem at hand is an all-to-all broadcast problem, the proper selection of the communication model is important. For example, a simple broadcast from each peer to all others periodically would provide good freshness, but would heavily overload the network.

In Gossip, each peer periodically talks to some other peers in the network. Gossip protocols are much more scalable, since each peer does not communicate with everyone else but to a few selected peers. Gossip protocols do not also require as much synchronization as traditional reliable multicast protocols. The natural resiliency and graceful degradation of gossip protocols is also one of their attractions.

Multiple ways of doing gossip exist. In a static model, an overlay topology can be constructed over the underlying network, and each peer can convey its information to its neighbors according to the overlay topology, which propagate the information further to their neighbors. This way, hopefully, information is disseminated to everyone in the network. Dynamic Gossip schemes do not restrict the neighbors a peer can talk to each time, but is decided dynamically. We consider the dynamic model of gossip in our studies, as it avails more flexibility in choosing our neighbors. In the following section we describe the issues in dynamic gossip and also the various schemes, which we consider in our experiments.

Dynamic Gossip Schemes

As mentioned above, in Dynamic Gossip, each peer talks to some other peers periodically and communicates information to them. In the resource monitoring system, we consider that each peer maintains a database of resource information about all the other peers in the network. Each database record corresponds to a peer and stores its information like IP address, latency, the resource information and some other information (related to gossip schemes), which we describe later. The idea is that each peer in the network has a "fairly good" idea about the current characteristics of *all* the other peers in the network. Of course, the value of "goodness" may vary for different peers in its database.

The Whom and What model

Every time a peer decides to gossip, it needs two pieces of information, which we term as the *whom vector* and the *what vector*.

Whom Vector
This vector contains the list of the peers to which the gossiping peer will talk to in that particular gossip instance.

What Vector
In our gossip model, whenever a peer gossips, it can give information about others also, besides itself. The *what vector* constitutes the list of items from its database which it will communicate to the peers in the what vector mentioned above.

Basically, the different gossip schemes can be differentiated in how they decide the contents of the *whom vector* and the *what vector*. Our work in this project focuses on this aspect, and tries to find out an effective way of choosing these vectors such that the resource information dissemination is efficient.

Gossip Schemes

For the following gossip schemes, two parameters *whom_n* and *what_n* are passed on to the schemes, which do gossip based on these. whom_n is the number of peers each peer talks to when it gossips. what_n indicates the number of records in the database which it sends to others. These factors decide the network load for the gossip schemes. If we keep this parameters constant for different schemes, then we can compare the different schemes keeping the network load constant. The gossip schemes basically differ in how they use these parameters to do the gossip.

Also each peer maintains the database of records mentioned previously in two sorted lists. First list is sorted according to the latency of the peers from itself. Thus the closer peers are in the beginning of the list. Second list is sorted according to the "freshness" of the information it has in its database. This is useful, if the peer wants to discriminate what to send to others based on the freshness of the information it has about others. These two ordered lists are called latency_list and freshness_list respectively in our further discussions.

Broadcast

What Vector: All the records in the database are included in the *what vector*.

Whom Vector: All the peers in the topology , we know about currently are included in the *whom vector*.

This is basically a simple all-to-all broadcast model.

Random

What Vector: A peer selects what_n records randomly from its database. Its own record is always included in the what vector.

Whom Vector: A peer selects whom_n peers randomly from its peer list. It makes sure that it does not include itself.

Top

What Vector: A peer selects the top what_n item from its freshness_list. The idea is to propagate only the freshness information it has from its database.

Whom Vector: A peer selects the top whom_n peers from its latency_list. It only informs the closer peers around it about the information it has.

As we shall see, this results in a division of topology into overlapping zones, where each peer talks to other peers within the zone centered around itself.

Spatial

This is a more involved scheme, which we consider. A theoretical model for Spatial gossip was proposed in [1]. Here the idea is to choose the *whom* and *what vector* in an exponentially decaying manner. The probability that a peer is chosen for gossip is exponentially distributed based on how close it is. Similarly for freshness of data.

This model is slightly complicated because there are various ways of doing this exponentially distributed selection of peers and records to communicate. [1] suggests the model that probability of communicating with a particular peer is $P = c_x d^{-r}$ where d is the distance metric (latency or freshness in our case), r is a constant between 1 and 2 and c_x is a normalization constant. The intuitive idea is to talk to near ones more frequently than those further away.

This does not keep constant the number of peers we talk to, each time we gossip. Also, si nce d is the distance metric, it is not rank based, i.e. if a peer has a freshness_list with 50 items but their freshness is low compared to another's peers freshness_list, its what_vector will be poorly populated if we use the above scheme, compared to the other peer. Therefore various types of spatial schemes can be considered like metric based or rank based.

Bin-Halving

We have developed another gossip scheme, which is based on the rank rather than on the distance metric. This scheme is called Bin-halving.

Consider the selection of the whom_vector. From our latency_list we need to select whom_n peers, which follow a spatial distribution. We divide the latency_list into bins of constant size b. The first b items are in the first bin, next b items in the second bin and so on. Then for the whom_vector, we randomly select peers from each of these bins, such that the number of peers we select from each bin decays exponentially. Also the sum of these needs to be equal to whom_n. All these constraints lends itself to a mathematical formulation of the problem:

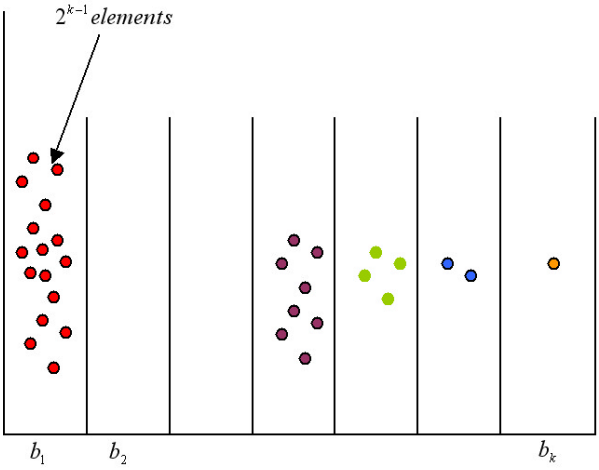
Let the size of the latency_list = n
whom_vector size is fixed = whom_n

We need to decide the number of bins k ($(b_1, b_2, ..., b_k)$ of equal size, such that if we choose 2^{k-i} peers randomly from the bin b_i, then the sum $2^0 + 2^1 + ... + 2^{k-1} = w\text{hom}\_n.$

Solving the above we get

$k = \log_2(w\text{hom}\_n + 1)$

Thus we have k bins of size $\frac{n}{k}$ each.



The bin-halfing scheme

In the bin-halving scheme, the size of the what_vector and whom_vector can be kept constant but its distribution can follow a spatial scheme.

Experimental Results

In this section we describe the experiments, which we conducted to study how the different gossip schemes fare with respect to each other.

The Gossip Metrics

First of all, we describe the metrics we use to compare different schemes. Discussion of these metrics is important as "freshness" can be defined in various ways and depending on the application a particular metric may be more preferable than the others. Some metrics, which we study in our experimental results, are:

Network resource Usage related

Messages sent: This measures the total number of messages sent by the peers in the network. Each record which is passed on to another peer is counted as a message.

Bytes Sent: This basically measures the load on the network in terms of total bytes sent.

Freshness Related

Measuring freshness is slightly more involved and different schemes could be compared in different ways according to different metrics.

In our model, each peer updates its resource information periodically. Whenever it updates its resource information, it increments its *heartbeat* by 1. Therefore, each resource record is associated with a peer and has a particular heartbeat value.
In our simulations, we define the measure of freshness of a particular record to be equal to the number of subsequent updates from the originator of that record or the *heartbeat_difference* between the record we have and the current heartbeat of its originator. Higher this *heartbeat_difference*, more stale the information is.
Note that in actuality, this *heartbeat_difference* cannot be measured by a peer to gauge the freshness of the records it has since it doesn't have access to the current heartbeat of every other peer in real time.

However since we have a simulator, we can use this metric to measure the freshness, since we can directly compute this difference. There we sometimes also use the term *oracle_freshness* to describe this metric.

We have three metrics based on this:

Average Heartbeat Difference:

We average the *heartbeat_difference* of all the records a peer holds in its database and then average this value over all the peers. This gives us an idea of what is the average freshness of the information peers are having about each other. We also measure the standard deviation in this, to see how the average *heartbeat_difference* varies across the peers. We see later, that different schemes have difference values for this deviation, depending on what is their *what_vector*.

Heartbeat difference distribution:

This is a bar chart, which shows what is the distribution of the heartbeat difference in the database each peer holds. If the value of heartbeat_difference=0,1,2 is high then it the information is more fresh. This gives us an overall idea of which scheme provides better freshness.

Fresh Item count:

This is an important metric as it measures the useful information a peer has in its database. Information is useful if it fresh. We can set various boundary lines for this freshness. In our simulations, we count an item as fresh, if its heartbeat_difference is <=k where k is a bound on freshness which may be decided according to the application. In our simulations we fix this value to 2.

A peer may have information about lots of peers, but only some of that information may be actually useful.A P2P application may just refer to the useful records for particular tasks.

Simulation Description

In this section, we describe our simulation approach to evaluate the different gossip schemes according to the metrics described above.

Simulator Design

We developed a discrete event Peer to Peer simulator in C++. A peer is modeled as an object which can receive and set simulation events. We have 3 kinds of events in our simulator:

1. SEND: On arrival of this event, a peers initiates a gossip instance in which it populates the what_vector and whom_vector according to the gossip scheme. Then for each peer in the whom_vector, it sets RECEIVE events at current_time + latency between itself and the other peer. It also sets a new SEND event for itself.
2. RECEIVE: This event indicates arrival of a message from another peer. The peer which receives this event, updates its database with the new information in an ordered fashion (in the order of both latency and freshness, as discussed previously)
3. PROBE: This event initiates metric collection. Periodic PROBE events are inserted into the event queue, which calculate and output the metrics for the entire P2P system.

Maintaining Freshness_list

We previously said that a peer cannot actually know the heartbeat difference of the records it has in its database from their originating peers. Then how does it decide which information in its database is fresh and how does it maintain the ordered freshness_list ? It uses another internal metric for this which we call *decay_freshness*. It measures how old the information is according to time , not the number of heartbeats. At time of origin, the *decay_freshness* of the message is 100. Each time a message is sent from one peer to another. the *decay_freshness* decreases by a function of the latency of the link. The peer which receives the message, is responsible for this. This at each RECEIVE event, the receiving peer recalculates the decay_freshness of the message by:

new_decay_freshness = decay_freshness - f(latency between itself and the sending peer).

This is a heuristic of freshness of resource information, which each peer can use to estimate the freshness and give preferential treatment to its database records in terms of freshness. We see that this heuristic indeed works in practice, and schemes, which exploit this internal metric, result in a lower heartbeat difference for the resource information.

Topology Generation

To compute the latencies between the different peers, we need to construct a latency matrix which can give us the latency between any pair of peers. We needed a realistic topology to construct this latency_matrix. We used the BRITE topology generator to generate a “Router Topology” with the following parameters:

Parameter	Value
Powerlaw Model	Waxman
alpha	0.15
beta	0.2
HS	1000
LS	100
m (Number of links added per new node)	2

After generating the topology, the latencies were computed using Dijktra's algorithm for all pairs and the latency matrix was constructed.

We generated topologies of sizes 256, 512 and 1024. In our simulation, we currently use a topology of 512 nodes and 1024 edges.

Strategy

We described various dynamic gossip schemes earlier. We have two different vectors which determine our gossip: *whom_vector* and *what_vector*. The different schemes can be applied to the two vectors independently. That is, we may choose the Random scheme for the whom_vector and Top scheme for populating the what_vector. So various combinations can be considered each having its own properties. In our simulation studies, we try to study these combinations and try to come up with a good way of deciding which scheme is good for each vector.

We keep the *whom_n* and *what_n* parameters constant and then try different combinations and then see which scheme performs better under the same conditions of network load.

We also vary the *whom_n* and *what_n* parameters to see how these affect the schemes. All schemes would show improvement, but some schemes may benefit much more from increasing these parameters and hence may be better in terms of scalability.

Results

All the discussion here is for experiments performed on a 512-node graph.

We define a tuple as follows:
(whom_algorithm whom_n, what_algorithm what_n)

what_n – size of *what_vector*
what_n – size of *whom_vector*

Comparing the various *what* strategies

1) Comparing the simulation results for *(random 10, random 10)* and *(random 10, top 10)*

Convergence of the database size [Graph 1] is slower for the top scheme than for the random scheme because every node is only giving out information of their close neighbors. Fresh item count [Graph 2] for the top scheme is about 50% better than the random scheme (13). So, in using top instead of random, we have improved the quality of information transferred for the same bandwidth consumption. Also, the heartbeat difference distribution [Graph 3] is shifted to the left for the top scheme when compared to the random scheme.

2) Comparing the simulation results for *(random 10, random 20)* and *(random 10, top 20)*

In this case, we have increased the number of items sent in each data transfer from 10 to 20. As a result, we get much faster convergence [Graph 6]. The Fresh Item count [Graph 7] with the top scheme is really high! This is around 45. The corresponding increase in the random scheme is 8%. So we find that doubling the number of node's information sent in case of top increases the fresh item count by almost 100%!

3) Comparing the simulation results for *(bin-halving 10, top 10)* vs. *(bin-halving 10, random 10)*

Here we are comparing the different *what* strategies with *bin-halving* as the *whom* strategy. Just like in random whom strategy, the binning whom strategy showed similar increase in the fresh item count [Graph 12] when we switched from a random what to a top what strategy. The increase was from 15 to 26 (over 70%). This increase is much more than the improvement we saw when using random whom strategy (was 50%). The average heartbeat difference [Graph 13] is quite similar in both the cases. The deviation of the average heartbeat difference has a lot more variance in the case of the top scheme. This requires further analysis.

We clearly see that deciding what to send based only on the basis of our estimated freshness improves the results drastically. We also see that as the size number of items sent is increased, we see a huge increase in the case of our top scheme, while the random scheme showed only marginal benefits. The only thing where the top scheme "lags" behind is the time needed for convergence. But this is not a very good metric for most purposes. As we have

demonstrated, the quality of information is improved (the freshness count etc) by using the top scheme. So we effectively get a somewhat narrower, but highly improved (in the sense of freshness) view of the world using the top scheme. As seen from the example of increasing the number of items to send, the top scheme scales much better than random. This needs to be further examined using some other metrics (future work).

Comparing the various whom strategies

Since we have established that the top scheme for what is much better than the random scheme, we fix what to top in this section, and try the different whom schemes.

1) Comparing the simulation results for *(random 15, top10)* and *(bin-halving 15, top 10)*

Convergence [Graph 16] is better in the random scheme (600 vs 1300). Again this is expected, and not necessarily bad for the bin-halving scheme for most applications. The fresh item count [Graph 17] was about 30% better with the bin-halving scheme than with the random scheme. This is because of the exponential priority that binning assigns. It prefers nodes that are closer to ones that are at a distance. This improves the quality of information, since nodes learn more often about nodes close to them. This is a very elegant distribution where every node has fresh information (the top what scheme), and information about its neighboring nodes (the bin-halving whom scheme).

Initially it was expected that the average heartbeat difference [Graph 18] case of bin-halving would be lower (better) than random. This was not found to be so, and can be explained in the following way. Since each node talks primarily to its neighbors, and rarely to nodes that are far, nodes at a distance generally rarely hear about nodes that are far. And since there are more nodes that are far than the number of nodes that are close (and talk more often), the heartbeat difference increases. Another interesting thing to note is that the variance of the data is much greater in the case of bin-halving. This is again because the information about close nodes is generally very fresh, while that of distant nodes is quite stale. This is both desireable and expected (scales well!)

The heartbeat distribution graph [Graph 19, 20] is also quite interesting. Here we see that in the case of bin-halving, the number of nodes with low heartbeat difference (0-5) is consistently larger than in the random scheme. However, the bin-halving curve falls off steeply thereafter, but has a long tail. The long tail is because of the large number of nodes about whom information is not very fresh.

2) Comparing the simulation results for *(random 31, top10)* and *(bin-halving 31, top 10)*

We now increase the number of nodes to whom information is sent to 31. Compared to when information was sent to 15 nodes, the convergence [Graph 21] time is about halved for both bin-halving and random schemes. The fresh item count [Graph 22] increased to about 118 in the case of random from 45 (160% increase). Similarly, the bin-halving scheme also increases the number of fresh items from 60 to 150 (150%). The order of increase is quite similar, and this leads us to believe that both bin-halving and random scale equally well when the number of nodes to which we are talking is increased. But bin-halving still has a significantly higher fresh item count. We feel that this difference can actually be improved much more, and trying to improve the bin-halving algorithm is left as future work.

The average heartbeat difference[Graph 23] decreases just as expected by about the same mount in both the cases. Random is still better for the same reasons we gave in the previous case.

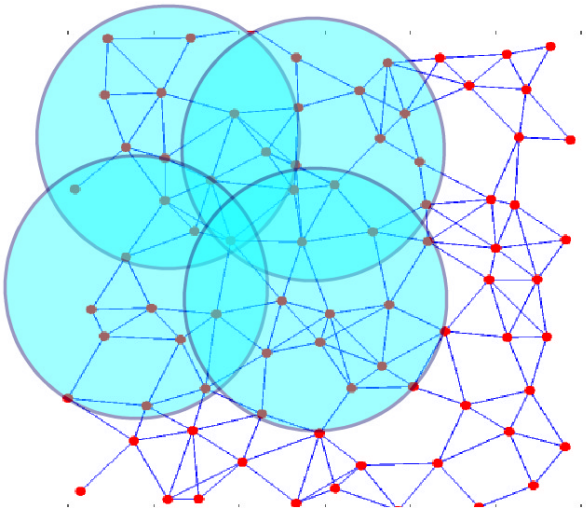
3) Comparing the simulation results for *(top15, top10)* and *(bin-halving 15, top 10)*

Using the top scheme for deciding *whom* to send information to gives quite interesting results as well. For starters, the top scheme database size never converges [Graph 26]. It stabilizes around the 200 mark, but never really goes beyond that. This is because the top scheme leads to the formation of overlapping islands, with nodes in an island only talking to others on its own island. So the number of nodes that can have their information transferred to a particular node is limited. This is not desirable for most monitoring applications. Our bin-halving approach is simply a heuristic to improve this.

The fresh item [Graph 27] count is 50. This is less than the fresh item count for the bin-halving scheme, where it was 60. Also, this is more than the count for the random scheme. Interestingly, the count was around 100 when the number of nodes to send information was increased to 31, which is less than the corresponding values for both bin-halving (150), and random (118). The reason for this is that in the case of top scheme, each node talks to a more or less fixed set of nodes in the steady state. This limits the number of nodes whose information would reach a node. In the case of the random scheme, a node talks to all nodes over a period of time, and even in bin-halving, it talks to distant nodes based on a priority model.

The average heartbeat difference [Graph 28] is a linearly increasing function. At time 5000, it was as high as 250. In Bin-halving has this had stabilized at around 8. The reason for this linear increase is again because of the island formation, with people not talking to far off people at all.

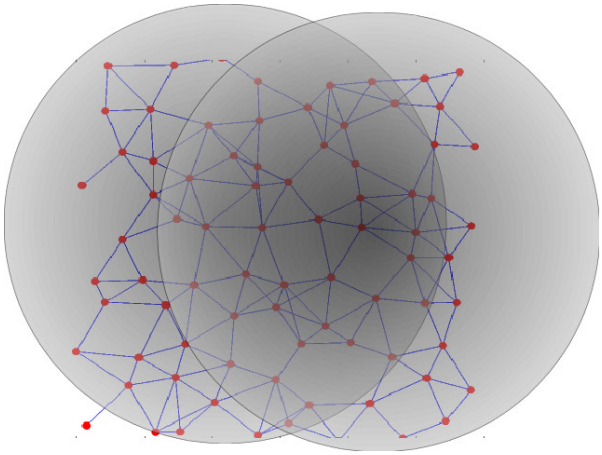
Intuitive Notion of Top and Spatial Gossip schemes



Top Gossip Scheme

In the top gossip scheme, the graph is divided into many overlapping zones, with each peer gossiping to the peers in the zone centered at itself. So within a zone, all peers will have fresh information about the peer at the center. This results in high freshness of information to the peers inside the zone, but only to a select few (those inside the zone). Each peer may know only about a fraction of the total peers in the system, as shown in the results above. Therefore its fresh item count is not very high, though the heartbeat_difference distribution is very good, with peak on the extreme left.

Spatial Gossip is like a combination of Random and Top schemes. It combines the advantages of both to provide a higher fresh item count than the both, though faring somewhat poorly in the average freshness of the system. We take advantage of sending information only to people close to us, resulting in higher freshness of information being sent to them. However, a peer doesn't know only a select few peers. Since we also send to distant peers, this results in more popularity of a peer's information in the network, and a higher fresh item count, as it combines both random and top properties into one.



Spatial Gossip Scheme

Conclusions

With the goal of designing an efficient P2P resource information dissemination system, we have focused on finding an efficient communication model for propagation of this information with network resource scalability and freshness of information as two primary concerns. We studied various combinations of dynamic gossip schemes using simulation (including a rank based spatial gossip scheme called bin-halving) and found that depending on the metric, different schemes may be preferred.

One of the main parts of our work was design of meaningful metrics for representing “freshness” and then evaluating various schemes with respect to these.

For choosing the *what_vector*, we found that the TOP gossip strategy comes out to be the best, which seems intuitive. Sending only fresh items to others results in better freshness of the entire system.

If Fresh Item Count is the primary metric, bin-halving along with top (for *what_vector*), proves to be the best strategy.
For Average Heartbeat Difference as the metric, Random Gossip scheme for the whom_vector works the best. Bin-Halving results in a larger average, with a much larger standard deviation.

Future Work

The scalability of these schemes needs to be studied. We mainly compared the schemes to see their relative performance. But network load may be a major bottleneck in limiting freshness of data and it needs to be studied more extensively.

We studied just one formulation of the Spatial Gossip strategy. Spatial Gossip can lend itself to various types of formulations and many others exist. Bin-halving may not be the best and better schemes may exist like metric based schemes instead of rank based.

A theoretical analysis of some of these schemes would help further validate our conclusions.

Also, more realistic topology needs to be generated which more closely resemble the current end-to-end topologies in the Internet. Router model may not be the best, as it doesn’t have any hierarchical structure.

References

[1] David Kempe, Jon Kleinberg, Alan Demers: Spatial Gossip and Resource Location Protocols. *Proceedings of STOC 2001, Crete, Greece.*

[2] Luisa Gargano, Adele A. Rescigno, Ugo Vaccaro, "Communication Complexity of Gossiping by Packets", *Journal of Parallel and Distributed Computing*, vol. 45, pp. 73-81, 1997.

[3] J-C. Bermond, L. Gargano and U. Vaccaro, "Fast Gossiping by Short Messages", *SIAM J. on Computing*, 27, 1998, pp. 917 - 941.

[4] A. Czumaj, L. Gasieniec, and A. Pelc Time and Cost Trade-Offs in Gossiping**,** *SIAM Journal on Discrete Mathematics*, Vol. 11, No. 3, pages 400-413, August 1998.

[5] M.Chrobak, L.Gasieniec, W.Rytter, *Fast algorithms for broadcasting and gossiping in radio networks*, 41st Annual IEEE Conference on Foundations of Computer Scienc, FOCS'00, Redondo Beach, 2000, pp. 575--581.

[6] S. Hedetniemi, S. Hedetniemi, and A. Liestman. *A survey of gossiping and broadcasting in communication networks.* Networks, 18:319--349, 1988.

[7] Robbert van Renesse, Kenneth Birman and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. ACM Transactions on Computer Systems (TOCS), November 2001

[8] PlanetP: Using Gossiping and Random Replication to Support Reliable Peer-to-Peer Content Search and Retrieval". F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. *DCS-TR-494, Department of Computer Science, Rutgers University.*