

CS497 Research Report

Timothy Jacobs

November 24, 2008

1 Introduction

Here I describe the simulation of a Simple Churn-Tolerant Structured Peer-to-Peer Scheme layed out in [2]. The scheme guarantees certain properties to hold within the network, such as logarithmic diameter, logarithmic searching, etc., with high probability. The properties are founded and guaranteed on probabilistic expectations, so a simulation of the network using these probability models is needed to get a better picture of how the network will react in practice.

The Simulator is written in Java 1.5 to mimic a network that runs for some time t with stable network size N . It is simple in the fact that the simulation of all nodes is done in serial. The network loops for t cycles, each adding a sequence of new peers, then removing peers who have stayed for their predetermined length. Every so many cycles, the network is inspected to determine varying statistics, e.g. diameter, average degree.

2 Simulation and Network Structure

The network of peers(or nodes) is built upon a "backbone" graph. The network scheme [2] allows for a wide variety of "backbone" graphs; the Simulator presented here builds upon a Cube Connected Cycle graph. The network is set up with an expected stable network size, N , and a runtime length, t . The dimension r of the CCC graph to build the network on is then determined by $r = \log(N/\log^2 N)$ [2]. The simulator is then looped for t cycles, each cycle composed of: removal of nodes whose time has expired, addition of new nodes, and calculation of network statistics.

The removal of nodes is simple; each is assigned a session length when they arrive based on the probability distribution which governs how long the node stays in the network. This length is decreased every cycle until it reaches zero, when it is removed as described in [2].

Based on analysis presented in the scheme [2], nodes arrive based on a Poisson distribution with rate λ . This is achieved by each cycle sampling a Poisson random variable (from a simple algorithm [3]) with rate λ and adding that many nodes to the graph, serially. Each node's session length, $1/\mu$, is sampled then from an arbitrary distribution, with mean $1/\mu$. Based on real world statistics in Stutzbach, et al. [5, 6], weibull, log-normal, and exponential distributions fit well to mirror actual peer session lengths. For most simulations, the session length was taken from random variable with weibull distribution, with shape parameter $k = .59$ (based on [5]) and varying the scale parameter λ such that the mean of the random variable will be $1/\mu$. The Poisson variable rate λ and weibull mean $1/\mu$ are then chosen so $\lambda/\mu = N$. The node is then added to the network as described in [2].

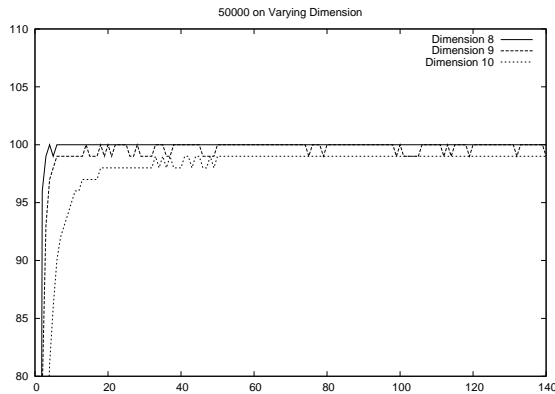


Figure 1: Coverage

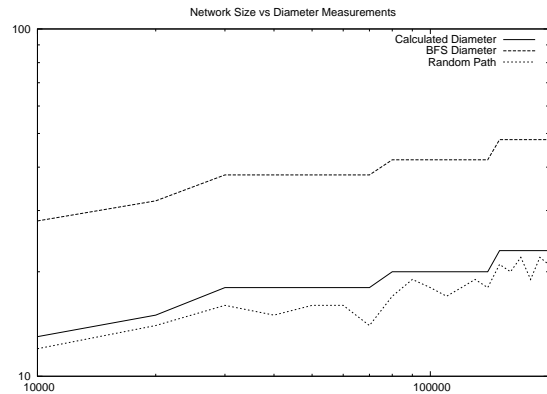


Figure 2: Diameter

3 Network Statistics

The simulation keeps track of the basic network statistics: diameter, average degree, as well as those of interest to this specific network construction: vertex coverage, i.e. the percentage of vertices in the "backbone" network that are covered by network nodes; average coverage, i.e. the average number of nodes covering a vertex in the "backbone"; and random path length, i.e. the average path length through the network over $\log(N)$ paths.

3.1 Coverage

The coverage of the network is important, without 100% coverage routing through the network cannot be assured to be done efficiently, and if the coverage becomes too low, the network may become disconnected. Coverage is measured as the percentage of vertices in the "backbone" that are covered by a node in the network, as explained in [2]. Coverage is tied closely to the dimension of the CCC graph, in relation to the number of nodes in the network. If the dimension of the CCC graph is too large for number of nodes, the network can never reach 100% coverage, as seen in the dimension 10 network in Figure 1. However, it can also be seen that if the dimension fits the number of nodes, the graph will reach 100% coverage quickly. The dimension $r = \lceil \log(N/\log^2 N) \rceil$, with N being stable network size, has provided 100% coverage once the network reaches stable size with every simulation.

3.2 Diameter

The diameter d of a CCC graph of dimension n can be computed by $d = 2n + \lfloor n/2 \rfloor - 2$ for $n \geq 4$ [1]. The diameter of the network was computed approximately by traversing the network with breadth-first search, and taking the diameter to be twice the height of the produced tree. This provides a reasonable estimate, within a constant factor. With networks of a large number of nodes, this becomes to inefficient in terms of time, in many simulations, tripling the run time. A faster approach is to consider the random path. In the random path, two nodes would be pulled from the network at random, and a path would be routed between them. $\log(N)$ paths are sampled, and their lengths averaged together. This allows a much faster measurement of the networks diameter. As seen in Figure 2, the random path actually provides a much more accurate diameter than the breadth-first search; due to the efficient shape of the CCC graph, the BFS-diameter is greater by almost a factor of 2.

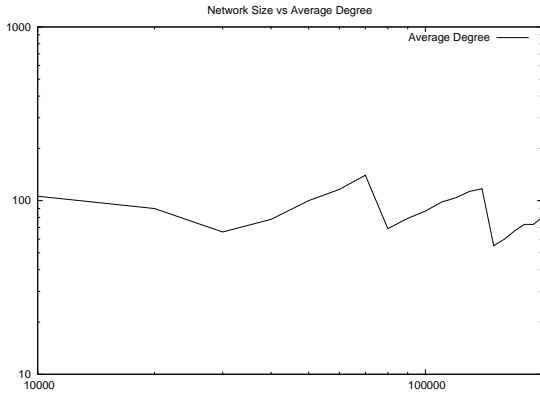


Figure 3: Average Degree

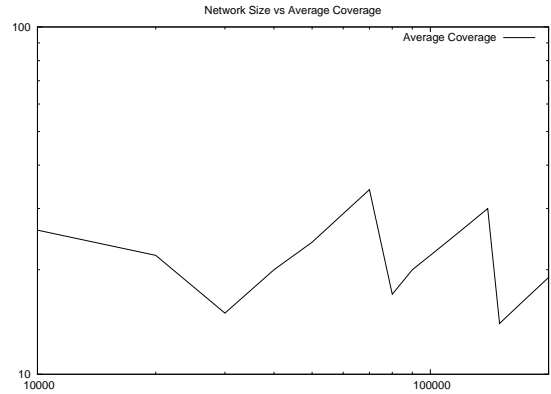


Figure 4: Average Coverage

3.3 Average Degree and Coverage

The average degree of a node in the network can be seen in Figure 3 to grow with the network size, keeping within a constant factor of $\log(N)$. The sharp drops in the average degree occur when the network size is large enough to support a higher dimension CCC graph, spreading the nodes in the network over many more vertices in the backbone CCC graph.

The average number of nodes covering a vertex is heavily related to the average degree of a node. As seen in Figure 4, it follows the same pattern of growth. It is interesting to point out, from networks ranging of size 10000 to 150000, the average degree and coverage stay constant, around average degree of 100 and average coverage of 25.

4 Changing Dimensions

The network will not stay at constant size forever and must compensate for drastic changes in network size. To accomplish this, the dimension of the network must be increased or decreased to adjust for an increase or decrease in overall network size. This will be accomplished in as decentralized process as possible, so that each node must work to keep track of the network stability, while gaining a dynamic, fully decentralized peer-to-peer network.

4.1 Method of Detection

Each time unit in the simulator, a node in the network runs a simulation method mimicking normal operations of a node. At regular intervals, the will look into its neighbors in the network and attempt to ascertain if the current network is stable, then take measures if it is not stable. The regular intervals were tested with success at 100 to 500 time units; any shorter and the fluctuation of network would interfere to greatly for one individual node to correctly calculate the network status.

The node determines whether it is stable by using the average degree of several nodes, and checking if it is close to the ideal stable degree, D . The degree of the nodes stays within $O(\log N)$, and based on previous network simulations of nodes in the networks up to 1000000 nodes, the stable degree will fall around 100, ± 50 , as seen in Figure 3. Each node tracks the progression of the average degree of a sampling of nodes in the network, called A . If A , begins moving away from the stable degree size, the node will then lower its

dimension if A is falling or increase its dimension if A is rising. There is a buffer of ± 65 around D , so that random variations in the sample average degree will not trigger incorrect dimension change.

Each dimension change will only decrease the dimension of the node by 1. This is to prevent the network from growing or shrinking too quickly. If the nodes of the network dimensions would make large increases in dimension, the nodes would need to expand to too large a CCC, increasing the time the network is disconnected. Decreasing the dimension greatly would cause the cycles of the CCC to be shortened too much, causing difficulties in network routing.

4.2 Changing a Node's Dimension

A node will change dimension relatively simply. For each node of dimension r in a CCC, its id is $\langle w, i \rangle$, with $0 \leq w < 2^r$ and $0 \leq i < r$. To decrease the dimension of a node to dimension $r - 1$, the r th bit of w is set to 0, and if $i = r - 1$, i is randomly set to either 0 or $r - 2$. Once each of the nodes undergoes this, it will collapse the network to a CCC of dimension $r - 1$. To increase the dimension of a node to $r + 1$, a random bit is added to the $(r + 1)$ th of w , and if $i = r - 1$ or $i = 0$ the nodes have an equal chance of retaining their i value or setting $i = r$. With this, the CCC is split into a CCC of dimension $r + 1$. When each node makes a change in dimension, they remove themselves from the network, then readd themselves to the correct position, using their previous neighbors to smartly reconnect them with a minimum of messages sent.

4.3 Propagating the Network Change

If each node was left to change by themselves, many nodes would not get a chance to change or change too slowly and leave the network unstable or disconnected. To remedy this, once a node detects a network instability and changes dimension, it sends a message to each of its neighbors, suggesting for them to change their dimension to its new dimension. Each node monitors these messages, and once it receives enough of them (simulations have shown that around 5 is sufficient to eliminate any false positives), it will change its dimension to the suggested dimension, regardless of their own measure of the average degree. As each changes, it sends its own suggestion messages, which will then effectively propagate the change in dimension across the network.

As the network is undergoing change, nodes are still joining it, so their dimension is decided by rounding the average dimension of all nodes that share its vertex and node id. Since all nodes that share a vertex have the same degree, once change to the dimension comes to a vertex, they will all change very quickly, so the new node will either have the new correct dimension or be in a vertex that the change has not propagated to yet.

4.4 Conclusion

Figure 5 depicts a typical network response to a large change in network size. The network was simulated for 450000 time units, where a 50000 node network dropped to a 20000 node network. The network is forgiving in small decreases, but once the network drops too far at $t = 610$, the network corrects its self quickly, 70% of the nodes in the network switching in less than 7000 time units. Due to nodes continually being added while the network is adjusting, perfect instantaneous convergence to the new dimension is unlikely, but as the network progresses, it will continue to self-adjust and reduce the average dimension in all of its nodes to the new correct dimension. The random path statistic in Figure 5 shows the average of a sampling of nodes route lengths when trying to reach a random assortment of nodes, mimicking requests during normal network operations. While the network contains nodes of differing degrees, it is still able to function normally, with few disconnects or routing problems.

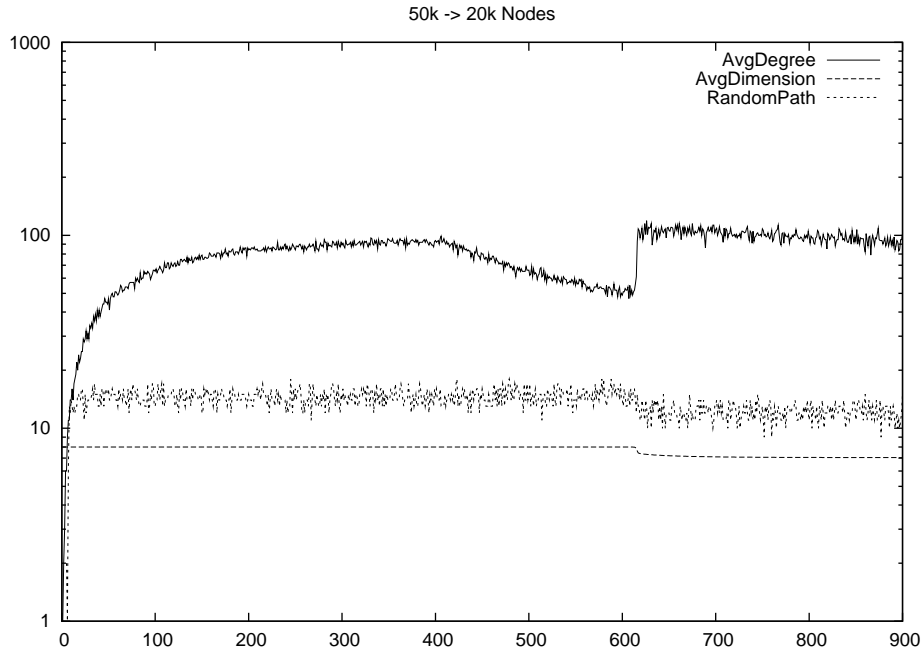


Figure 5: Dimension Adjustment

5 Further Investigation

Now the network model has shown that it should behave fairly well in a real-time, real-life peer-to-peer network situation, but the network is still highly theoretical. Many more inquiries into the practical development and optimization of the network are needed before it can be realistically deployed. One such venture for optimization is into the use of the erasure codes outlined in Efficient Dispersal of Information [4]. The erasure codes can be used to eliminate some of the redundancy in the network as such.

When a file, F , of size N , is uploaded to the network, it is split into k pieces, each of size m . These pieces are labeled $S_i, 1 \leq i \leq k$. Then, an n is chosen, so that $n > m$, and n/m is close to 1. Then, create an $n \times m$ matrix A whose rows a_i are linearly independent so it is non-singular. Then, F_i , for $1 \leq i \leq n$, are constructed as such: $F_i = (a_i * S_1, a_i * S_2, \dots, a_i * S_k)$, so F_i is a vector of length k . These F_i pieces are what will be stored by the network - they are now created in such a way that any m of them can be used to fully reconstruct the file F .

A note on how F is then reconstructed: Any unique m of the F_i pieces are retrieved - matrix C is created then with F_i as its rows in order. Then the $m \times m$ matrix A' is created by taking rows a_i from the original matrix A if F_i is in the list of pieces retrieved. A' is then inverted. Then $F = S_1, \dots, S_k$, where $S_i = (A'^{-1})(c_i)$, where c_i is the i th column of C .

Now, how data insertion and retrieval could take place in the peer-to-peer network. When data is inserted, it is mapped to a vertex, divided into the n pieces, F_i , specified above. If there are N nodes in the network, the vertex will have approximately $M = \log N$ nodes covering it. To achieve storage sizes equal to the original plan, each node could simply store a random m pieces. A large savings in the individual and overall storage is then achieved if each node stores less than m pieces. So if there are $l < m$ pieces stored

in each node, when another node requests the data, it only needs to contact m/l nodes to retrieve the file, provided the nodes all have distinct pieces. The pieces then must be divided amongst the nodes so that at least m unique pieces (ideally all n) remain in the network through its churn.

One way to help insure this is when dividing the pieces, is to divide the file into a prime number of pieces - then distribute each piece to the nodes in a circular fashion - assign the first piece to the first node, second piece to the second node, etc, then repeat until each node has l pieces. Choosing p to be carefully will allow few nodes to be repetitions of others.

Another way to insure this would be to have the nodes query the status of their files in the network if they would leave the network gracefully. If the node leaves gracefully, it can probe the other nodes on its vertex for information about the frequency of the pieces, and, if the frequency of its pieces is low, can send its pieces to a selection of other nodes before it disconnects to preserve them.

When a node A retrieves a file, then, it routes to the node B which should have the data as specified in the network outline. Once it contacts B, A can retrieve all of B's pieces of the file it needs. Those will not be sufficient, so then B can then provide A with its neighbors who cover its vertex. A can then use those neighbors to retrieve the rest of the m pieces, and reconstruct the file. This easily forces each vertex to share the load of storing and sending files across all of its vertices.

5.1 Conclusion

The modeling of the network can be greatly improved with a more realtime, concurrent system, which will allow a more accurate representation of how the network will react to real-time stimulus, and how the network can be adapted to current needs of peer-to-peer networking.

References

- [1] I. Fris, I. Havel, P. Liebl. The Diameter of The Cube-Connected Cycles. *Information Processing Letters* 61, 1997. Elsevier.
- [2] G. Pandurangan and S. Jagannathan. A Simple Churn-Tolerant Structured Peer-to-Peer Scheme.
- [3] Donald E. Knuth. *Seminumerical Algorithms*, The Art of Computer Programming, Volume 2. Addison Wesley.
- [4] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, Vol. 36, Issue 2, 1989.
- [5] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger On Unbiased Sampling for Unstructured Peer-to-Peer Networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement*, 2006.
- [6] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement*, 2006.