Emergent Behavior through Local Decision Making in a Peer-to-Peer File Sharing Network

Authors Redacted

CSCE 475H: Multiagent Systems
Department Of Computer Science and Engineering
University of Nebraska – Lincoln
256 Avery Hall, Lincoln, NE 68588-0115 USA

Abstract

In this paper, we present peer-to-peer file sharing networks in the context of a multiagent system environment, for the purpose of studying how the local decision making functions agents employ lead to global coherence in performance and structure for the entire network. We used the Repast Simphony agent-based modeling software package to simulate a peer-to-peer file sharing network, and run a number of experiments within this environment to determine the relationship that structure, connectivity, churn rates, and initial knowledge have with network efficiency. After running many trials of these experiments, we present the compiled results and analyze the effects and outcomes we observed. While the desired emergent behavior resulting from varying the underlying network structure was not observed, we did identify a number of significant factors that play a role in overall network performance. As a result of this analysis, we conclude that node connectivity and churn rates are important local characteristics that have substantial effects on network efficacy. Finally, we conclude the paper with discussion of the implications and applications of our results, as well as future work we believe could yield important discoveries in this space.

Commented [LS1]: Good.

Introduction

The introduction of the ~~internet~~Internet fundamentally changed the nature of computing, and offered new ways to communicate, interact, and share data. Today, the traditional client-server model is the most prevalent method by which computers interact over the ~~internet~~Internet, a centralized model of network communication that relies on dedicated machines to broker requests and hold information. An alternative model is peer-to-peer networking (P2P), which forgoes the centralized server architecture and relies instead on ad-hoc communication directly with other computers in the network topology. Popularized (in the eyes of the mainstream public) by companies like Napster, peer-to-peer networking has been, and remains, an important facet of the computing landscape. Although the infamous Napster music sharing website is long defunct, peer-to-peer file sharing networks live on today through applications like BitTorrent and GNU Gnutella. These software applications enable nodes in the network, also known as peers, to access content such as books, movies, music, or other digital wares, by searching other connected nodes in the network.

Thus, peer-to-peer enjoy a number of unique advantages over the centralized model. First, peer-to-peer networks are, by construction, decentralized, and are generally not vulnerable to the loss of a single (or even multiple) nodes. Since copies of a file most likely exist on multiple nodes in the network and an individual peer will only host a relatively small number of files, the loss of individual nodes will typically not impact the ability of other peers to successfully retrieve files. In contrast, if a centralized download server goes down, all requests for files will fail. Second, peer-to-peer networks allow vast quantities of data to be stored and be readily accessible to the network, without imposing massive storage requirements on a centralized server. The storage cost of hosting the universe of files available on the network, or corpus, can be spread out among millions of connected peers, rather than on a small subset of specialized content servers. Additionally, this prevents central servers from becoming a bottleneck in the system. Third, and perhaps most importantly for their continued prominence, peer-to-peer networks are commonly used to transfer illicit goods (such as copyrighted movies or music), as they enable anonymous participation in the network and do not require an easily seized centralized server to host all of the illegal content. To summarize, peer-to-peer networks provide a robust protocol for file sharing that offers decentralized control, is more resilient to poor network conditions, and gives nodes access to a vast corpus of data.

Commented [LS2]: Good summary.

However, centralized client-server network topologies have emerged as the dominant file sharing protocol, primarily due to their simplicity, centralization, and efficiency. First, nodes do not have to utilize complex handshakes or communication protocols to request information across a distributed graph of peers; rather, they can query a single server node. The simplicity of this transaction, for both clients and servers, makes it an attractive choice for many practical scenarios. Second, centralized servers do not suffer from the same fragmentation of data and knowledge that plagues distributed networks. The reduced complexity in file distribution across the network by hosting them in centralized nodes makes it simple to quickly make files available across the network. Finally, and most importantly, client-server architectures have proven to be the more efficient, and thus economical, choice for most common scenarios. While peers in a P2P network typically rely on commodity consumer hardware (which suffer from poor performance and limited ~~internet~~Internet connectivity), even modest investments in server hardware can create powerful nodes capable of maintaining thousands of concurrent connections, a more economical solution than linearly scaling the number of peers.

While peer-to-peer file sharing networks have a number of unique features that make them attractive for use, centralized architectures also have many desirable properties that increase the overall efficiency of the network. By altering the communication protocols of individual nodes, peer-to-peer networks can induce some of these benefits in efficiency while maintaining their fundamental peer-to-peer structure. In this paper, we first present a multiagent system that simulates a simple distributed peer-to-peer network for file sharing. We then explore various network configurations and local decision preferences that induce greater efficiency in the network as a whole, while preserving the underlying peer-to-peer structure. In the multiagent system, nodes employ local value functions to maximize the speed of each query transaction (Local Decisions), and we explore the impact on network performance, as measured by high average query speeds across the graph, optimally distributed traffic, and low failure rates (Global Coherence).

Commented [LS3]: Okay.

Multiagent System Design

To conduct these experiments, we have designed a multiagent system that models a distributed peer-to-peer (P2P) file sharing network. In order to ensure our system accurately models real world peer-to-peer networks, the mechanism design draws heavily from standard network architectures and common P2P software applications like Gnutella. The environment in which agents interact provides simple network connectivity and addressing protocols, much like the ~~internet~~Internet. The decisions individual agents make within the system simulate them running a common P2P software package like Gnutella or Gnutella2, which provide a framework for how agents should structure themselves and communicate with others in the network.

Environment Design

The public ~~internet~~Internet relies on addressing schemes, like IPv4 and IPv6, to assign unique identifiers to all ~~internet~~Internet connected devices, and provides routing infrastructure that ensures any two connected computers can communicate knowing only each other's addresses. In a similar fashion, our multiagent system environment assigns each agent a unique number that serves as its "IP address," and provides mechanisms for agents to directly address other agents in the system using only this IP address. Just like the IPv4/IPv6 schemes, there is an upper bound of possible addresses known to all agents in the system; any address in that range is valid, and could (but does not have to) correspond to an agent attached to the network. When an agent sends a request to another peer, one of two things can happen. First, if the request could be received by the destination agent and responded to appropriately. Alternatively, if the destination node is busy serving other requests or if no destination node exists with the given address, the request will time out. This behavior models closely how typical request-response interactions play out between ~~internet~~Internet ~~-~~connected computers.

Commented [LS4]: Incomplete sentence. Grammar.

While the network routing infrastructure is an important central piece of our system's environment, the way the total set of files in the system, or the corpus, is handled is another aspect of the environment. As a simplifying measure, all files are identified with a number, and the maximum possible file identifier is known to all agents. This simulates agents having the knowledge of the existence of all files in the corpus so they can effectively query for files they desire. In practice, a subset of files in the corpus are distributed to nodes in the network; thus, files may exist on one or more nodes, or not at all.

Agent Design

Part of the beauty of this peer-to-peer system is that there is only one fundamental type of agent: a peer (also referred to as a node). Peers are computers connected to the ~~internet~~Internet that are participating in the file sharing network, and share a number of common behaviors. Peers may enter or leave the network, request files from others in the network, and help propagate file queries throughout the network. While there is only one agent type in this multiagent system, peers can be highly customized by altering their configuration, which leads to very different agent behaviors. Thus, our system often exhibits multiple classes of peers that share common configurations. Each class of peers can be defined by a certain permutation of the following characteristics:

Connection Limit

An agent's connection limit indicates the number of active connection/interactions it can process simultaneously. Agents maintain a local queue of requests (both incoming and outgoing), which represents the work each agent needs to do. Each tick in the simulation, agents can process up to their connection limit of these requests at once. As a result, the connection limit is perhaps the most important differentiating factor for peers. The connection limit is an important result of our mechanism design, which models real world hardware scenarios. Agents with low connection limits, modeling lower end computer networking hardware, are only able to connect to a few other nodes at a time, and thus are typically found on the edge of networks. Conversely, agents with high connection limits model higher end hardware, such as high performance servers, which can serve many other nodes at once and often live in the center of networks.

Request Rate

One of the most important agent configurations, the request rate controls how often a node requests a new file from the network. This models the different types of nodes in the network: some may frequently request content, while others may exist primarily to serve as middlemen (and thus infrequently request new files).

Starting Files

Each agent begins the simulation with a set of files it already possesses. The goal of the multiagent system, therefore, is to ensure nodes requesting those files are able to traverse the peer-to-peer graph to find them. Throughout our experiments, we vary the number of files that each node begins with to model different behaviors.

Ping Limit

When agents connect to the peer-to-peer network and start requesting files, they will initially have no knowledge of other nodes in the network. At this time, they use one of their fundamental actions, pinging, to search for other peers in the network. The ping limit controls how much time agents will spend pinging other nodes in the network when they do not have enough information about the network topology. For nodes near the edge of the network, low ping limits are appropriate as they only need a few connections into the peer-to-peer network. Nodes more central to the network's structure may be configured with a higher ping limit, as they have a greater desire to learn about nodes in the network.

Commented [LS5]: Okay.

Churn Rate

The churn rate controls the rate at which active nodes drop out of the network, and at which inactive nodes rejoin the network. This models both the effect of changing network conditions (which may cause nodes to lose connection with the network), as well as the natural tendencies of agents' participation in the peer-to-peer network. Churn rates for agents near the edge of the network may be relatively high, while those near the center may tend to be low.

Learning Rate (Alpha)

The learning rate influences to what extent nodes incorporate the results of recent interactions with the network into their beliefs of the system. This parameter gives us as system designers the ability to control the process by which agents learn about the network, which we can use to test various hypotheses.

Common Configurations

Throughout our experiments, we consider two important classes of agent configurations: leaf nodes and ultra-nodes. These classes have distinct characteristics that impact their behavior, leading to interesting network interactions. The experiments utilize various structural settings, where nodes are either configured identically or divided between these two classes, to test the impact of differing structures and configurations on overall network performance.

Leaf Nodes

Leaf nodes make up the majority of the network, and exist primarily to gain access to other files. Consequently, they are characterized by low connection limits and ping limits, but high request rates, churn rates, and starting files. Leaf nodes are named because they exist at the edges of the peer-to-peer network, and will often join the network to request a file and subsequently leave. In real world scenarios, leaf nodes are very similar to individuals who may use Gnutella or BitTorrent to download music or a video, but do not remain connected to the network constantly.

Ultra-Nodes

In contrast, ultra-nodes primarily serve the role of facilitators. Their chief motivation for their actions is to gain as much knowledge about the system as possible. To enable them to effectively act as middlemen, ultra-nodes typically have a very high connection limits and ping limits. However, ultra-nodes have very few files, rarely request new files, and rarely leave the network (low churn rate).

Agent Actions

Each agent has the following actions available to it:

- Ping: Discover peer i
- Pong: Reply to a ping from peer i
- Query: Request file f
- Query Hit: Reply to a query for file f
- Join: Join network N
- Leave: Leave network N (die)

Pings and Pongs are what generate the network. Instead of designing the network to conform to a predetermined structure, the team decided to let the network develop as it would in the real world,

Commented [LS6]: Good. Openness in MAS.

Commented [LS7]: Clear. But not quite about "decisions". For example, how would an agent decide whether to reply or not reply, or when to reply, etc. how would agent decide when to leave or join the network?

where peers gradually learn about other peers over time. This also allows us to observe the structure that emerges overtime. Queries and Query Hits are what allow the network to actually carry out its function of file sharing. Join and Leave allow for the very real possibility that peers would decide to stop participating in the network or that peers would decide to start participating in the network over time.

Agent Learning

In order to facilitate fast query speeds, each agent maintains a belief system about the network and adapts its beliefs over time. In our simulation, agents are primarily concerned with accumulating three types of knowledge: (1) knowledge of the existence of other nodes, (2) knowledge of the existence of other files in the network, and (3) knowledge of the speed with which other nodes are able to obtain files.

Learning in the two existential knowledge settings is fairly straightforward because the knowledge is fairly constant over time. For example, once an agent, i, knows that another node, j, has a particular file, f, then i can count on j always having file f. Additionally, if node i knows that node k exists, then node i can be fairly confident that k exists at a later time, except in the rare circumstance that k has dropped out of the network. Because of the relative simplicity of the knowledge, agents are able to immediately incorporate it into their belief system without any heuristic to process the new knowledge. In practice, this means that when an agent learns of the existence a file, it immediately adds that file and its owner to a table. In our simulation, agents learn of files by being involved in a successful query, in which case, based on a configurable environment setting, they will either learn that the next agent in the chain is capable of getting the file or that the final agent in the chain actually has the file. In addition, when an agent learns of a new node, it adds that node to a separate table along with an initial belief about its performance. Agents are able to get this information about the existence of nodes either through the ping/pong mechanism described above or through being involved in a query, in which case the agent learns about all the nodes after it in the call chain.

Learning in the context of node speed is more complicated because the speed at which a node is able to fulfill a request is highly dynamic, varying based on the current size of the nodes request queue and the specific file that is being requested. Consequently, the agents use a more sophisticated method for integrating new knowledge. Each time an agent is part of a query chain, it gets the timing information for each node that follows it in the chain. This timing information for each agent is the amount of time it took between (1) when the request was added to its request queue and (2) when the request was terminated, either by a query hit or by a timeout. If the file was found, the agent updates its belief about each node i for which it gets timing information to be $B_i' = \alpha * t_i + (1 - \alpha) * B_i$, where B_i' is the updated belief, α is the learning rate, t_i is the timing information, and B_i is the previous belief about the speed of node i. If the request times out, the agent updates its beliefs about node i with $B_i' = \alpha' * TIMEOUT + (1 - \alpha') * B_i$, where B_i' is the updated belief, α' is $\alpha' = \frac{t_i}{TIMEOUT} * \alpha$, $TIMEOUT$ is the environmental timeout constant, and B_i is the previous belief about the speed of node i. This reduced alpha helps ensure that nodes do not get punished too much for receiving a request right before the timeout. Using the timeout constant instead of the timing information for the node helps ensure that nodes do not get rewarded for receiving a request right before the timeout.

This accumulated knowledge gets used by the agents to help make the decision of where to send a request. If an agent knows that another agent has the file being requested, it will automatically send the request to that agent. If it knows that multiple agents have the file being requested, it will send the

Commented [LS8]: Okay!

Commented [LS9]: Okay!

Commented [LS10]: Hmm … a heuristic fix. Okay.

request to the agent that has the file and that it thinks is the fastest. If an agent does not know where the requested file is, it will send the request to the node that it believes is the fastest. These simple rules help the agents use their accumulated knowledge to increase the probability of a successful query and decrease the time it takes for the file to be found.

Commented [LS11]: Okay!

Commented [LS12]: Nice touch!

Desired Emergent Behavior

The desired emergent behavior of this system is that the nodes in the network will structure themselves such that the network provides high average query speeds across the graph, optimally distributed traffic, and low failure rates. Intuitively, for a file-sharing network to be successful, the network must collectively be able to find the requested files quickly (high query speeds and low failure rates). However, our team has identified an additional goal of getting balance among nodes of the same class, ultra or leaf. This balance will help the network cope with times of high volume and will help distribute knowledge of the network broadly across the different nodes.

Experimental Setup

In our multiagent system, peers employ local value functions to maximize the speed of each of their query transactions. Ultimately, these local decisions impact the emergent structure of the network as a whole (or lack thereof). A high performing network is characterized by high average query speeds across the graph, optimally distributed traffic throughout the network structure, and low failure (timeout) rates. Here, we desire to study how the local decisions agents make impact the global coherence of the network. To that end, we have established a number of hypothesis that pose significant questions about the relationship between these local decisions and emergent global structure. Finally, to test these hypotheses, we have implemented our multiagent system design and run various experiments.

Hypotheses

Here, we present a set of hypothesis that we wish to test using our multiagent system to better understand how various network structures and configurations impact overall network performance. We expect that varying parameters for local decision making will induce emergent behavior in the system, which we can study to identify how local decisions can lead to global coherence.

Hypothesis 1

A network that consists of nodes configured identically to both serve and request files at moderate levels will result in a stable network configuration that is not optimally fast.

Hypothesis 2

Configuring a network such that its nodes have significant distinctions in attributes, that is, there are clear leaf and ultra-peer nodes, will result in an effective network that achieves high query speeds and ultra-peers with significant knowledge of peers and file locations.

Hypothesis 3

A network configured with nodes that are very limited in possible concurrent connections will result in a network with poor structure and slow query speeds.

Hypothesis 4

A network configured with a high churn rate, that is, nodes will enter and leave the network frequently, will result in a network with poor structure, high failure rates, and slow query speeds.

Hypothesis 5

A network configured with nodes that have increased knowledge of the existence other nodes leads to a network where leaves are unable to effectively connect to high-knowledge ultra-nodes, and request speeds will suffer.

Experiment Design and Implementation

To test these hypotheses, we have implemented our multiagent system using Repast Simphony, an open source agent-based modeling system. Using this software package, we were able to simulate a peer-to-peer file sharing network with full control over system configuration. Our implementation gave us as system designers full control over the environmental design and agent decision making process by exposing a set of fundamental system characteristics as parameters within Repast. Using Repast's batch simulation tool, we were able to run a large number of simulations using various parameterizations in order to test our hypotheses.

Commented [LS13]: Good.

Repast exposed a wealth of data that let us better understand the performance of our system during each trial. First, we are able to visualize the network as the system progresses, depicting both how requests propagate through the network and what connections nodes have learned about over the course of the simulation. Second, we are able to graph aggregate information about system performance as the simulation proceeds, giving us an indication of timeouts and fulfilled queries in the system. Third, we dump both aggregate and non-aggregate data to a text sink, giving us highly detailed information about each node's performance during each tick of the simulation. This gives us access to data such as:

- Average Fulfillment Time
- Number of Fulfilled Requests
- Number of Timed Out Requests
- Number of Known Files
- Work Queue Size
- Node Status (Dead or Alive)

Unfortunately, there are a few limitations in the raw data provided by Repast. First, before the system has reached a stable state, aggregate data is misleading as many nodes have not yet started requesting or serving files. To combat this, we only consider data after 1000 ticks have elapsed in the simulation. Second, Repast is unable to automatically provide aggregate data for differing classes of nodes. Thus, we must manually analyze the non-aggregate data dump from each simulation. These data dumps, coming from the text sink in Repast, are very large files (multiple gigabytes in size), and we cannot directly import them into Excel. Thus, we have written a custom parser in C# to process the data and provide important statistics.

Commented [LS14]: Hmm … this is confusing to me. I don't understand "stable state", "have not yet started requesting or serving files" …

Results

To test our hypotheses, we ran a number of experiments against our multiagent system using the Repast Simphony batch programming model. Each test was repeated 30 times, and run for 10,000 ticks in the simulation environment. This was chosen to ensure we had enough samples from which to derive statistically significant results ~~from~~, without incurring high resource costs (as each experiment run generates large quantities of data that must be stored and processed). In each run, 100 nodes were simulated in an environment with 1,000 total files. When the leaf and ultra-node experiment setting was used, 10 percent of the nodes were ultra-nodes. The global timeout was set to 100 ticks; if a request was not completed within 100 ticks, it would be automatically cancelled.

Throughout these experiments, we seek to understand how local agent decisions impact the global network performance and emergent structure (if any). To that end, we have identified several significant criteria for measuring the efficiency, performance, and structure of the network. First, we measure *network performance by observing the average fulfilment time for nodes*. This metric looks at the average number of ticks it takes for nodes to retrieve files they have requested. Intuitively, other things being equal, a better network will exhibit lower average fulfilment time. Second, we measure *network efficiency by comparing the number of fulfilled requests to the number of requests that timed out*. Ideally, well-performing networks should exhibit a stable and high fulfilment percentage, indicating that the network is able to serve requests reliably. It's important to note that the timeout percentage will never fall to zero; the simulation is designed such that it is possible for peers to request valid files that no active node has, so timeouts are an expected part of this model by design. Thus, a measure of good system performance is that timeout rates stabilize and remain relatively low. Finally, we measure *network structure by computing the variance in work queue lengths for each node class*. In a network where traffic is distributed effectively, the work queue lengths of nodes of the same class (i.e. leaf or ultra-peer) should be relatively similar. Wide discrepancies in work queue lengths would indicate that certain nodes are being chosen more often than others in query request propagation, indicative of poor network structure. Thus, a measure of good network structure is low work queue length variances. Each of the experiments discussed in this paper measure their performance using these metrics, providing a common framework for measuring the impact individual changes have on global performance.

Commented [LS15]: Sensible metrics.

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

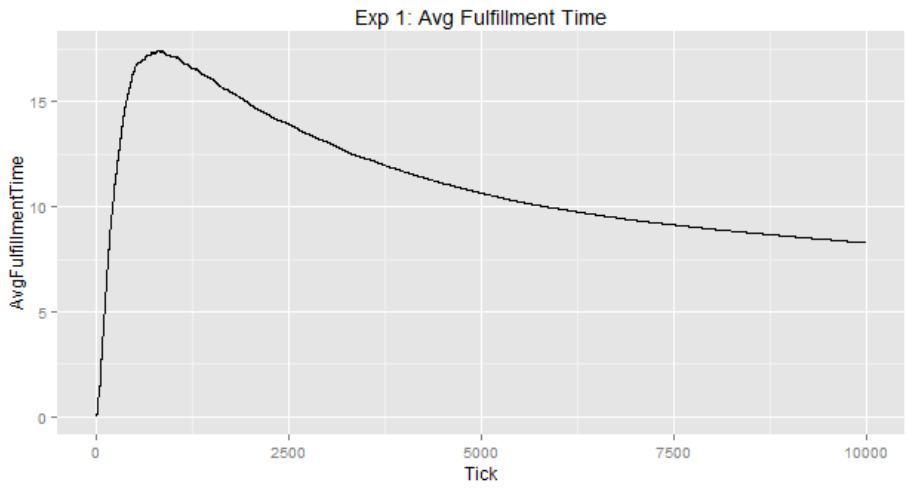
For each experiment, we first present our experiment setting and relevant parameters. Next, we note any unique observations about the system's performance under the test, and provide relevant comparisons to other tests where appropriate. Finally, we offer justification for why the observed behaviors emerged. This section presents the results of our experiments, and the conclusions section later in this paper provides important implications of these results on network design.

Experiment 1

Our first experiment is designed to test Hypothesis 1, and serves as a baseline performance indicator against which we judge the performance of other tests. In a sense, this experiment may be seen as a control, as it provides metrics for how a system will perform under typical configuration. For this test, each node was configured identically with values common to typical peer-to-peer nodes. Appendix A summarizes the full set of parameters used for this test, but the most important parameters used in this configuration are provided here.

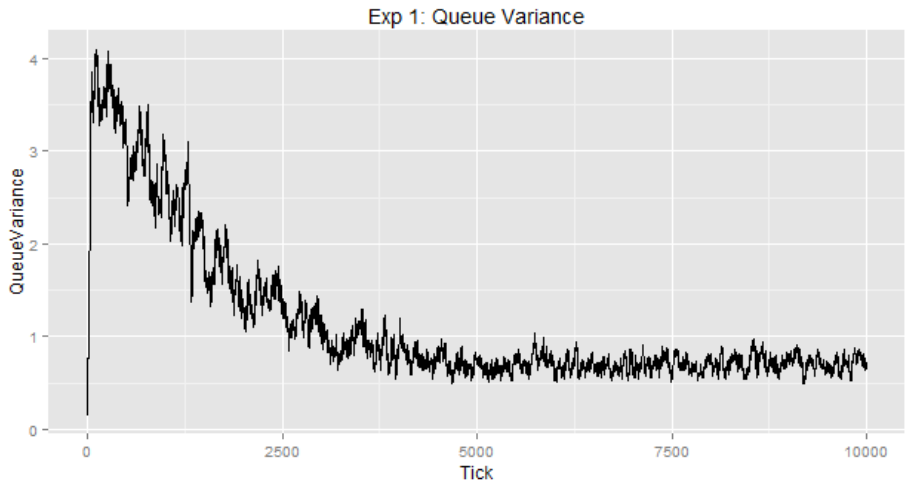
Parameter	Value
Request Rate	0.0075
Starting Files	125
Connection Limit	4
Churn Rate	0.00001

As seen in the graphs below, our baseline experiment performed well in all facets of our analysis. This experiment seemed to validate the learning algorithm that we implemented in the agents, because as time went on, the performance improved dramatically. This increase in performance can be seen both in the reduction of the average request time and in the reduction of the variance of the work queue sizes. Additionally, the number of successful queries far exceeded the number of timeouts, indicating that the network is efficient.



Commented [LS16]: Should have numbered your figures and tables. Figure 1, Figure 2, etc. Table 1, Table 2, etc. That way, much easier to refer to them specifically in the text.

Commented [LS17]: Numbers not shown. Hmm…

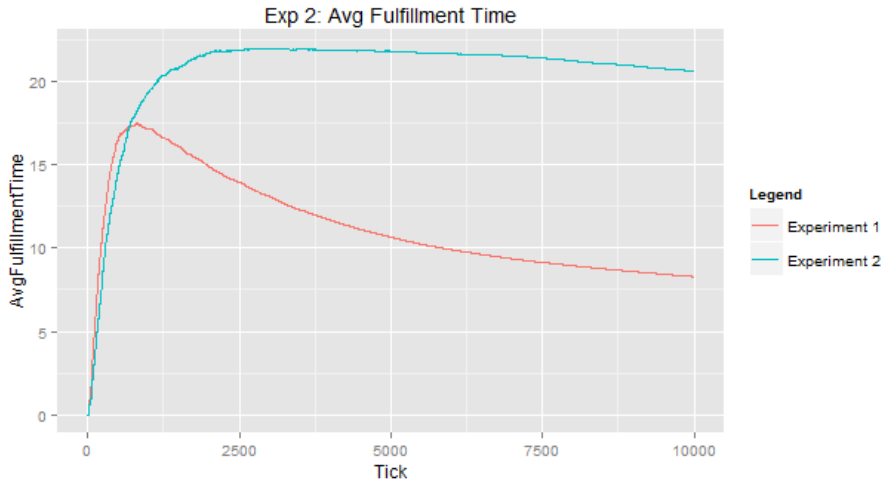


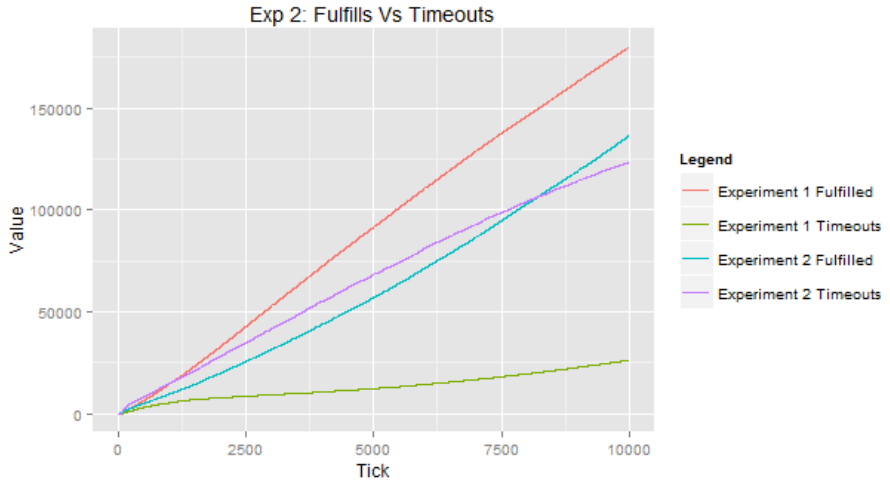
Experiment 2

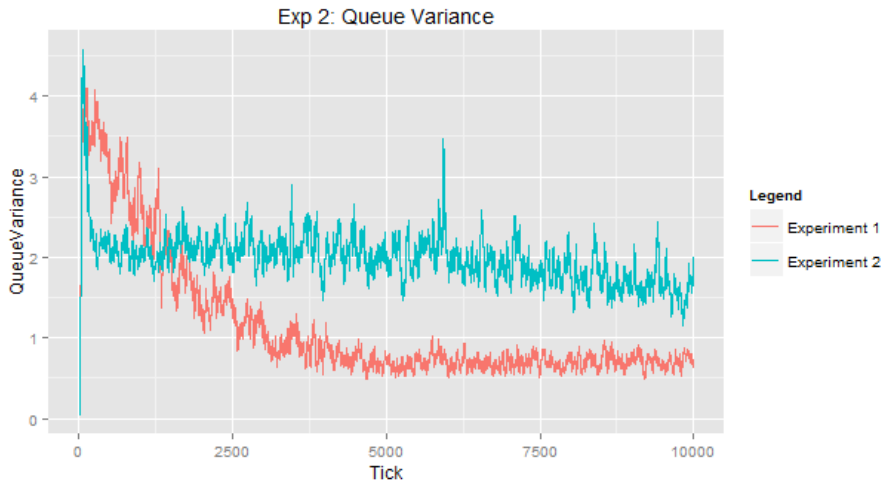
Our second experiment tests Hypothesis 2, which seeks to understand the impact of a different network structure on global performance. While each node was configured identically in Experiment 1, here we consider two classes of nodes: leaves, and ultra-peers. These two classes were configured with different parameters in several important areas, which gave them unique capacities and roles within the network. Again, the full set of parameters used in this experiment are listed in Appendix A, but the most important parameters are listed here.

Parameter	Leaf Nodes	Ultra-Peers
Request Rate	0.0075	0
Starting Files	125	0
Connection Limit	2	100
Churn Rate	0.00001	0.00001

While we hypothesized that the introduction of high performing ultra-nodes would increase the performance of the network, we found that the network in Experiment 2 actually performed worse on all three of our performance metrics. As seen in the graphs below, Experiment 2 had significantly higher average fulfillment times, significantly more timeouts, and far less traffic balance among the leaf nodes.


Exp 2: Avg Fulfillment Time
AvgFulfillmentTime
0
5
10
15
20
0
2500
5000
7500
10000
Tick
Legend
Experiment 1
Experiment 2


Exp 2: Fulfills Vs Timeouts
Value
0
50000
100000
150000
0
2500
5000
7500
10000
Tick
Legend
Experiment 1 Fulfilled
Experiment 1 Timeouts
Experiment 2 Fulfilled
Experiment 2 Timeouts



We believe that this deviation from our expected behavior was caused by a number of factors. In Experiment 1, all of the nodes had a connection limit of 3. In Experiment 2, while some nodes were made into ultra-nodes and given a higher number of connections, the leaves actually had their connection limit reduced to 2. This 33 percent reduction in the connection limit could have caused a significant amount of the reduction in Experiment 2 performance. Additionally, the reduction in performance could have been caused by an underdeveloped decision algorithm for the ultra-nodes. We observed that during the simulations, ultra-nodes would often pass queries on to other ultra-nodes. While this is not fundamentally bad (and, in fact, is in part by design), the ultra-nodes don't have any files and, therefore, can't be the final destination of the request. Thus, if requests never get sent out to leaf nodes, the request will inevitably time out. A third potential cause is that the learning takes longer with the ultra-nodes present, and the 10,000 tick limit was not enough to see the full benefits of the ultra-nodes. As can be seen in the "Exp 2: Fulfills Vs Timeouts" graph, the line for the number of fulfillments in Experiment 2 is concave up, while the line for the Experiment 1 fulfillments is very linear. Potentially, if given enough time, the number of fulfillments for Experiment 2 could catch that for Experiment 1.

Commented [LS18]: Why? Should not.

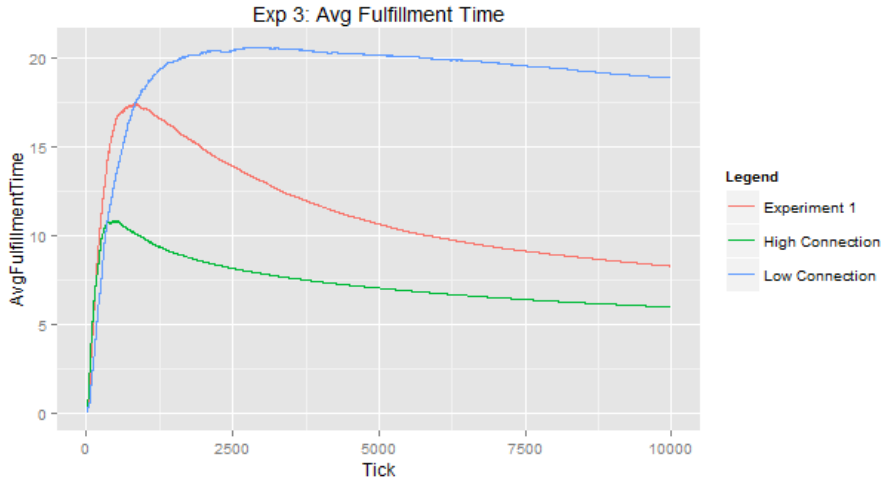
Commented [LS19]: Why?

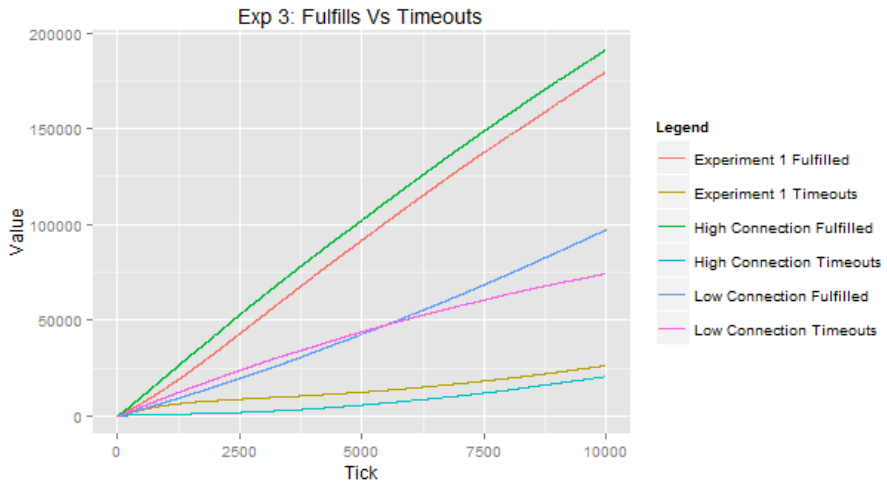
Experiment 3

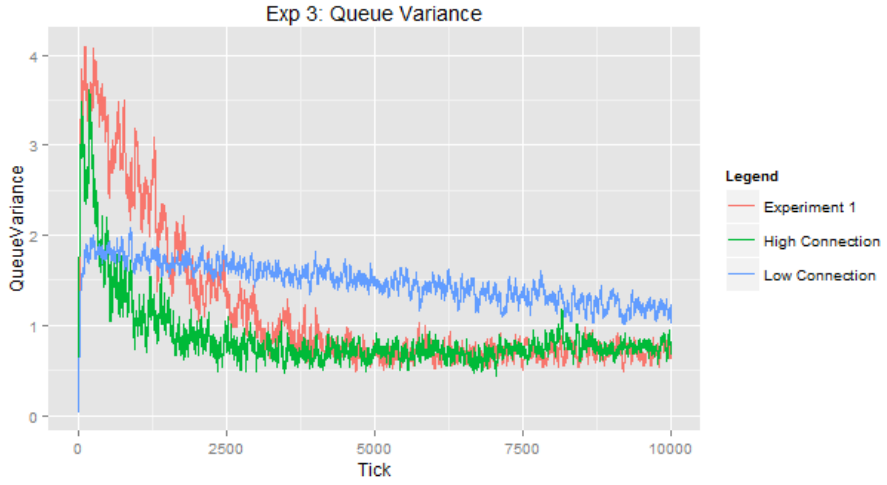
Our third experiment tests Hypothesis 3, which seeks to understand the impact changing node connection limits has on the performance of the network as a whole. To do so, we tested our design with a low connection limit, a medium (default) connection limit, and a high connection limit. All other parameters were identical to those used in Experiment 1, which enables us to accurately see the impact of connection limits without the influence of other changing parameters. As a result, the "medium" setting for this experiment is simply Experiment 1, and we ran our simulation 30 times for both lower and higher connection limits.

Connection Limit	Value
Low	2
Medium (default)	4
High	6

As seen in the following three graphs, the data from our experiments strongly backs our hypothesis that a higher connection limit will lead to better network performance. We found that the average fulfillment time was inversely related to connection limits, meaning that more connections lead to faster requests. We found that the number of fulfillments was directly related to connection limit and that the number of timeouts was inversely related, meaning that the highly connected networks were able to find a higher percentage of files. We also found that a higher connection limit tended to balance out a system more, although the control from Experiment 1, had a relatively similar equilibrium variance as the high connection experiment.







We believe that the experiments backed our hypothesis for a few reasons. First, when each node has a higher number of connections available, it can fan out the request more broadly, which will lead to a faster query time. Additionally, with a higher connection limit, each node can process more queries per tick, which will result in less time that queries have to wait in queues. Additionally, having more available connections allows the agents to learn more quickly, thus improving the performance.

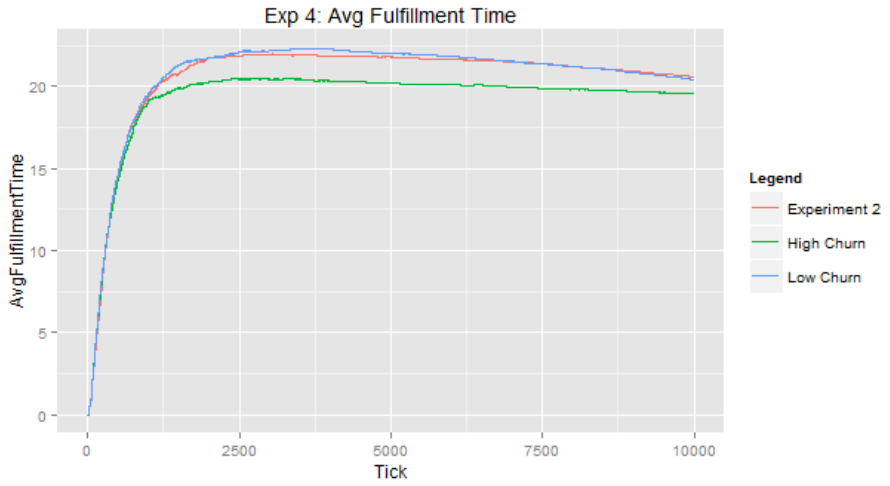
Commented [LS20]: Okay. Actually, more queries → more messages → more traffic congestion? But since traffic is not modeled here in the simulation …

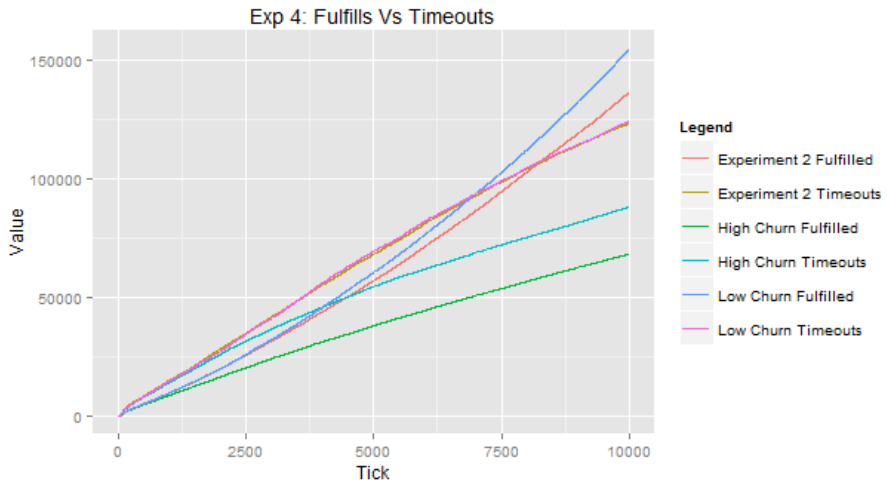
Experiment 4

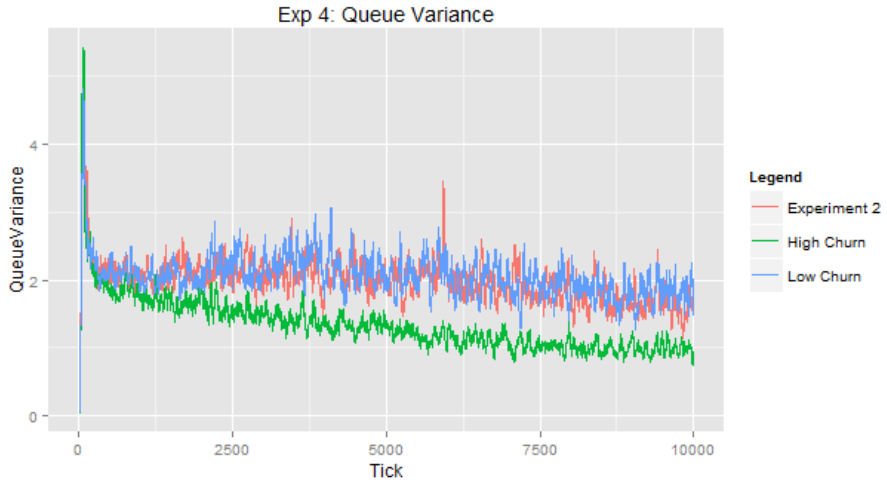
Our fourth experiment is designed to test Hypothesis 4, where we explore the impact that nodes entering and leaving the network ~~has~~ have on the stability and efficiency of the network. Much like Experiment 3, we tested this variable with a low, medium (default), and high churn rate. Because the churn rate is a much more significant factor in network topologies containing ultra-nodes (as they aggregate knowledge about the network), this experiment tests churn rate in the leaf and ultra-node setting, instead of configuring all nodes identically. As a result, while we still provide comparisons back to the results of Experiment 1 where appropriate, the results from Experiment 2 serve as a more appropriate baseline for this test.

Churn Rate	Value
Low	0
Medium (default)	0.00001
High	0.0001

We hypothesized that a low churn rate would increase the performance of the network, and although the low churn rate does not have the best performance for all of the metrics, we believe the data validates our hypothesis. As seen in the charts below, the low churn rate performs comparably to the Experiment 2 baseline in average fulfillment time, number of timeouts, and queue size variance. However, the low churn rate experiment significantly outperforms Experiment 2 when it comes to the number of successful queries. Not only is it always higher than that of Experiment 2, but it also seems to be increasing at a faster rate.







Although the high churn rate experiment seems to have the best performance when it comes to query speed and queue variance, we believe that it is actually the worst performer overall. It is by far the worst on what is arguably the most important metric, fulfillments vs. timeouts. Of the three churn rate levels, it is the only one that has more timeouts than fulfillments. If a file sharing network is unable to

successfully find files, then it is not worth having, regardless of its structure. Additionally, the fact that it has a slightly lower average fulfillment time is most likely caused by the fact that the fulfillment time does not include timeouts. In the high churn experiment, the requests that were fulfilled were most often the ones where one of the early agents knew where the file was located because in situations where the location is not known, the recently killed nodes will make it more likely that a random search will result in a timeout. This structure results in a lot of short request fulfillments but also very many timeouts, thus making the average fulfillment time metric deceptive. The variance of the queue size is also very deceptive because when nodes die, their queue size goes to zero. The high churn rate causes more nodes to die, thus causing a large number of queues that all have a size of 0, which reduces the variance. Overall, the data validates our hypothesis that the lower the churn rate, the better the performance of the network. This makes intuitive sense because as nodes die more quickly, the knowledge that the living agents have becomes less accurate.

Commented [LS21]: Okay. I am convinced.

Commented [LS22]: Good one!!

Commented [LS23]: Okay.

Experiment 5

Finally, our fifth experiment seeks to understand how the amount of knowledge individual nodes have about the network impacts global performance and structure. This behavior was controlled by the ping rate variable, which influences the desire of nodes to send pings into the network to learn about its structure. An increased ping rate will cause peers to desire more information about other nodes, and thus they will send more frequent pings to other nodes in the network when needed. In contrast, a lower ping rate indicates that a peer will be satisfied with only having knowledge of a few other nodes in the network. For similar reasons as Experiment 4, this experiment makes more sense performed in the leaf and ultra-peer setting, as leaves and ultra-peers have different motivations and desires to learn about the network. Thus, we again use Experiment 2 as a more appropriate baseline for this experiment, as it offers accurate comparisons against agent performance in each class.

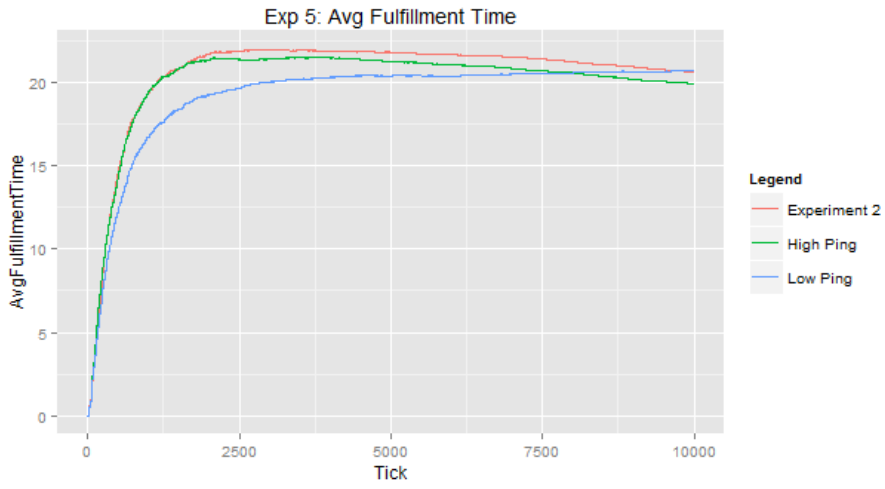
Ping Threshold	Value
Low	2
Medium (default)	3
High	5

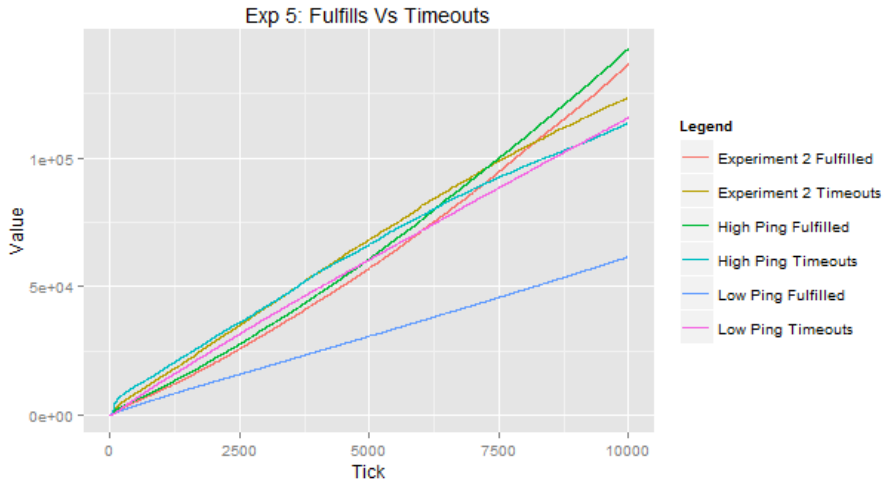
Although we expected that too much knowledge could actually hurt the performance of the network, we found that the higher the desire of leaf nodes to learn about the network, the better the network performed. Although the low ping experiment had the lowest average fulfillment time for most of the simulation, in the long run, the high ping and Experiment 2 lines pass it and continue heading downward. Additionally, its early lead in this metric was most likely due to it having a higher timeout probability, as discussed in Experiment 4. While the high ping experiment just barely edged out Experiment 2 with regards to number of fulfilled requests, the low ping experiment lagged far behind, ending up with significantly more timeouts than fulfilled requests. The variance was basically the same across all the experiments.

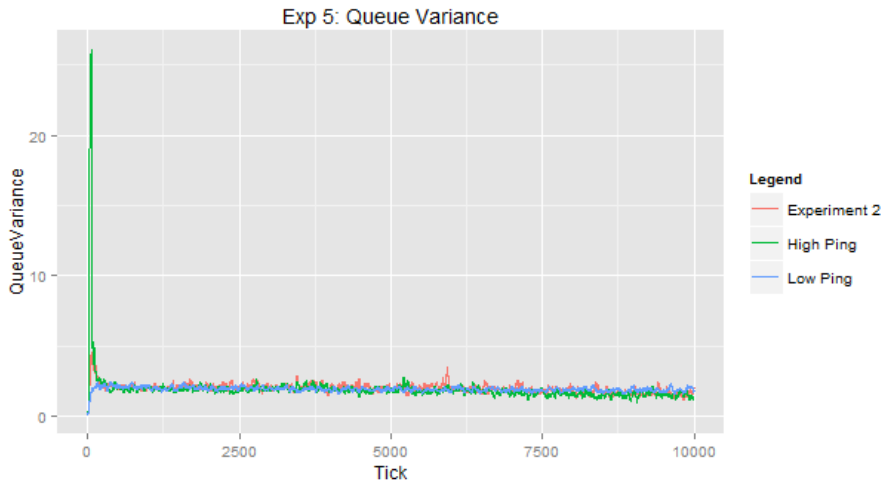
Commented [LS24]: !!

Commented [LS25]: Hmm … okay. But why did High Ping catch up after 7500 ticks? What caused High Ping to consistently improve?

Commented [LS26]: I am surprised by this just a bit. The Avg Fulfillment Time graph showed three very close performances. Would be nice to show the variance graph past 7500 ticks only , or around 6000 and 8000.


Exp 5: Avg Fulfillment Time
AvgFulfillmentTime
0
5
10
15
20
0
2500
5000
7500
10000
Tick
Legend
Experiment 2
High Ping
Low Ping


Exp 5: Fulfills Vs Timeouts
Value
0e+00
5e+04
1e+05
0
2500
5000
7500
10000
Tick
Legend
Experiment 2 Fulfilled
Experiment 2 Timeouts
High Ping Fulfilled
High Ping Timeouts
Low Ping Fulfilled
Low Ping Timeouts



We believe that the observed behavior was the opposite of our expected outcome for two reasons. The first is that we made the learning ability unlimited for all of the agents. The agents have no restrictions about how much information they can store and incorporate into their belief system. In practice, this situation would be fairly unlikely, as it would not be practical for an ordinary leaf node to store information about the whole network. The second reason is that they ultra-nodes were less active than we had anticipated, thus giving an exploratory leaf node the opportunity to amass a knowledge base similar to that of an ultra-node. We observed that the ultra-nodes were hardly more involved than the leaf nodes. Even though the connection limit of ultra-nodes was many times that of leaf nodes, their average queue size was only slightly larger. Our hypothesis was predicated on the belief that the ultra-nodes' high connection limits would make them a hub in the network, thus letting them collect knowledge at a far faster rate than any of the leaf nodes and making it detrimental for leaf nodes to send their requests to anyone but an ultra-peer. However, this did not play out in our simulation, and our hypothesis was invalidated.

Commented [LS27]: Okay. But why would storing this much lead to the observed trends/patterns?

Commented [LS28]: Numbers?

Commented [LS29]: If the ultranodes had done what you expected them to do, what would you have observed?

Commented [LS30]: The POJI here is missing some key logical steps connecting the asertions.

Discussion

Our experiment results provide interesting observations about the nature of peer-to-peer file sharing networks. While some aspects of the network performed as expected under our different tests, there were several tests where the behavior of the system did not validate our hypotheses. Here, we discuss the behaviors we observed and the implications our experiment results have on peer-to-peer file sharing network design.

Emergent Behavior

Interestingly, our experiments did not engineer the emergent behavior we were looking for in the leaf and ultra-node settings, and identical settings often produced higher performing networks. Reasons for this phenomenon are discussed both in our results and future work, where we justify some of the reasons this may be the case.

Regardless, we still observed somewhat unexpected emergent behavior with respect to increasing node connection limits in an identically configured experiment setting. Connection limits played an important role in determining the efficacy of nodes and the global network performance. When connection limits increased, we occasionally observed that particular nodes obtained more knowledge than their peers. In these cases, even without clear leaf/ultra-node distinctions, they naturally became significant intermediary nodes in the system. We believe that this behavior emerged for a few reasons. Increasing the connection limits gave the nodes the physical capacity to serve more requests, an important physical limitation that would normally prevent this behavior from emerging. Then, perhaps by chance, the node obtained relatively more knowledge than its peers during the simulation, and thus was able to fulfill requests more quickly. Since agents prefer to pass requests on to agents they know are fast, they would more often pick this node. This, in turn, provided the node with yet more information, creating a positive cycle of information and fast request handling that made them important to the network substructure.

Commented [LS31]: How much more?

Commented [LS32]: This is okay. But it does not explain why the ultra-nodes with high connection failed to become high-traffic nodes serving large requests.

Would have been nice to also plot graphs that show the average performances of leaf nodes vs. ultra-nodes.

Would have been also nice to include plot graphs that show the average performances of nodes with nodes sorted by their "knowledge".

Key missing element to more strongly support your POJI.

Applications

We believe that the results of this study yield important implications that can guide the design and development of future peer-to-peer file sharing networks, as well as provide a framework for measuring the performance of systems as a whole.

First, our approach of assessing network structure by calculating the variance of node work queues was an effective measure of overall performance and query distribution. This novel approach, not used in any of the prior work we consulted during our initial research phase, can be utilized by mechanism designers as an important network characteristic they can seek to optimize.

Commented [LS33]: Would have been nice to cite these references in your report.

Second, our results indicate that nodes with higher connection limits are better able to find files on the network (that is, they have higher utility). However, this also leads a significant increase in global network performance, a powerful indication of how local decisions made by agents can impact the system at a macro level. While this connection limit was specified via a parameter here for the sake of experimentation, in reality the number of connections a node chooses to maintain simultaneously is an important local decision an agent can make. For example, the popular P2P application Gnutella2 uses a protocol where leaf nodes maintain, by default, only 2 connections to the network. We believe that our results indicate that the connection limit is a significant local decision that agents (and the P2P software packages they run) can exploit to increase their own utility, as well as global system performance.

Commented [LS34]: Substantial.

Third, we believe that more intelligent handling of knowledge gained throughout the simulation can lead to better local decisions on who to pass queries on to. While our experiment results indicate that greater knowledge of other nodes in the network expectantly reduced network performance, we believe that enhancements to this algorithm could yield better performance. Our simplistic learning model only utilized a learning rate (alpha), and performed poorly when learning happened quickly at the beginning of the simulation. If agents were to adopt a more formalized reinforcement learning protocol, like Q-learning (which includes not only a learning rate, but also a discount factor), they would be able to better overcome the negative impact of this initial unhelpful learning.

Finally, our results confirmed our hypothesis that decreasing the churn rate for peers would significantly benefit the performance of the system. This was especially true for ultra-peers, which caused catastrophic drops in network performance whenever a highly central node lost connection with the

network. This indicates that the centralized model does have merit with respect to network performance, and keeping important nodes alive is an important consideration in network design. In environments where some degree of centralized knowledge is acceptable, investing in nodes who serve solely as middlemen can benefit the network. When these nodes are designed, it is important that they have the hardware capacity to serve many simultaneous requests, and that they do not desire to enter and leave the network frequently, as this causes significant drops in performance. However, if centralization is contrary to the goals of the system, system designers may find it beneficial to set arbitrary limits on the amount of knowledge nodes are able to accumulate, thus preventing centralized nodes from emerging. In that way, they may be able to insulate the network from detrimental effects if those nodes are lost, while only sacrificing modest amounts of network performance.

Commented [LS35]: Okay!

Future Work

Commented [LS36]: Okay!

If we were to make further iterations and improvements to our simulation and agent design, the first and most important thing we feel we would address would a be a fundamental re-evaluation of our agent design and the mechanisms used for agent interactions. As we alluded to earlier, we feel the learning techniques used for our nodes, notably the ultra-nodes, didn't adequately foster the type of behavior we were trying to elicit. In many cases, the emergent behavior that we did observe was contrary to what we were hoping to see. We still feel that there is strong merit to and value in exploring and researching mechanisms for ultra-nodes that would lead to emergent behaviors more in line with our hypotheses. Ultimately, we feel that while still a very valuable model, this initial finished iteration of our agent design didn't model the actions and logic of a real-world peer-to-peer file sharer as accurately as we had hoped for.

We think there is room to further experiment with many more parameter configurations. For our experimentations, we first established a moderately well performing set of baseline parameters using primarily trial and error, intuition, and observations made from simulation runs. For the five experiments we conducted, we left a large number of these parameters unchanged. Part of our reasoning for this choice was because we only wanted to manipulate the parameters that made sense to adjust for testing our original hypotheses. Another strong factor in this decision was simply the amount of time involved with creating and running the extra batch run configurations, and the large quantities of data it would produce. In future experiments, we would encourage creating a matrix of configurations that would encompass all possible combinations of the primary agent behavior-relevant parameters. Then, statistical analyses could be done to find relationships between the input parameters and the observed outputs. We feel this would be the best way to discover and analyze unknown emergent behaviors. Unfortunately, the computational overhead involved with processing the absolutely immense amount of data this would generate would be both very time consuming and challenging.

Conclusions

Peer-to-peer networks remain an important application of the ~~internet~~Internet today, offering a decentralized alternative for users wishing to communicate and collaborate digitally. Perhaps the most prevalent use of such peer-to-peer networks is seen in file sharing networks, where users connect over the ~~internet~~Internet to share movies, music, and more. Throughout this paper, we have discussed the use of such systems, and provided a framework for the design of a multiagent system to simulate such an environment. The Repast simulation designed in conjunction with this paper provides an interactive way to interact with peer-to-peer networks in a lab setting, where network configuration and node decision making can be manipulated to engineer highly performant networks and discover emergent behavior.

This paper presents five key hypothesis our team had regarding the impact of these parameters on network performance. First, we looked at a baseline network with "default" configuration, and observed that it exhibited behavior close to what we can observe in real networks. Second, we explored the notion of agent classes, where there are large differences in how sets of nodes are configured. Interestingly, although we hypothesized that this would increase network performance, our experimentation revealed the opposite. Justifications for this phenomenon are included throughout the paper, as we believe there are number of underlying causes, some related to immature system design that may have caused this. Third, we explored how node connectivity impacts the network. Validating our hypothesis, we found that there is a strong correlation between node connectivity and greater network performance, a fact that system designers can exploit. Fourth, we analyzed how nodes entering and leaving the network influences the accuracy of local knowledge and the efficiency of the global network. As we hypothesized, nodes leaving the network often caused a decay in information quantity and quality that had a negative impact on network performance. Finally, our fifth experiment looked at how agent learning rates impact global performance. Here, we saw an unexpected negative relationship between greater initial knowledge and eventual performance, which may be indicative of a number of underlying drivers. Regardless, we feel that applying more traditional reinforcement learning based systems would fundamentally change this relationship.

Peer-to-peer file sharing networks are a perfect application of multiagent systems, as the network is decentralized by definition and highly reliant on nodes cooperating to search for files in the network, a result achieved by making highly local decisions. However, as we have seen, these local decisions have important consequences on the performance of the network as a whole as measured by the efficiency, performance, and structure of the network, and are an important consideration for both system designers and agents themselves as they participate in peer-to-peer networks.

Commented [LS37]: Okay!

Appendix A: Experiment Configurations

	Base	Churn High	Churn Low	Conn. High	Conn. Low	Leaf & Ultra	Ping High	Ping Low
node_count	100	100	100	100	100	100	100	100
percentUltra	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
file_count	1000	1000	1000	1000	1000	1000	1000	1000
startingFiles	125	125	125	125	125	125	125	125
ultraStartingFiles	0	0	0	0	0	0	0	0
alphaRate	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
timeout	50	50	50	50	50	50	50	50
experimentSetting	identical	Ultra & Leaf	Ultra & Leaf	identical	identical	Ultra & Leaf	Ultra & Leaf	Ultra & Leaf
requestRate	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075
ultraRequestRate	0	0	0	0	0	0	0	0
churnRate	0.00001	0.00010	0.00000	0.00001	0.00001	0.00001	0.00001	0.00001
ultraChurnRate	0.00001	0.00010	0.00000	0.00001	0.00001	0.00001	0.00001	0.00001
connectionLimit	4	2	2	6	2	2	2	2
ultraConnectionLimit	100	100	100	100	100	100	100	100
pingThreshold	3	3	3	3	3	3	5	1
pingThresholdUltra	5	5	5	5	5	5	10	2

Formatted Table