Design and Implementation of a Decidable Peer-to-Peer File-Sharing Network

Dan Carpenter
Computer Science
Bemidji State University
ddcarpen1@cs.bemidjistate.edu

Abstract

A peer-to-peer network is one where data is transferred directly among many small computers, as opposed to using a large centralized server. Some peer-to-peer networks use peer-to-peer communication in combination with a centralized server that stores meta-data. This paper, however, is only concerned with pure peer-to-peer networks. "Decidable" is used to describe a network where it is possible to determine definitively whether a piece of data is--or more importantly, is not--on the network. This paper describes an implementation of a decidable peer-to-peer network called “Kuziva” that is under development.

Decidable Networks

With today's networks, bandwidth is the most limited resource. People have lots of hard-drive space and fast computer processors, so in designing a peer-to-peer network these considerations are not as important as bandwidth. When this paper uses the term "cost," it is only referring to the cost in terms of bandwidth. There are three major costs to consider in designing a peer-to-peer network: the cost of knowing what files are on the network, the cost of knowing what nodes (the term node refers to the fact that computers in a peer-to-peer network can act as both client and server) are connected to the network and the cost of searches. Note that the bandwidth used for actual file transfers is ignored.

The definition of a decidable network is a network where there is a formula to determine whether a file is or is not on the network. The only way to create a formula like this is to have an indexing system that keeps track of each file on the network. In a peer-to-peer network, this index is distributed across the nodes in the network.

Maintaining the index of files incurs the following costs. First, each file will have to be added to the index when it is put on the network and removed from the index when it is taken off. Second, since the Internet is unreliable and nodes are sometimes disconnected without warning, periodic checks are needed to make sure that files in the index are still available.

Fortunately the number of files added to and taken from the network is proportional to the number of nodes that index them. Therefore the average cost per node for maintaining the index of files remains constant.

Besides maintaining an index of files, there is a significant cost with just knowing which nodes are connected to the network. In the Kuziva network storing this information is the most significant cost.

At some point the network grows too large for a single node to keep track of each of its peer nodes. At this point the Kuziva network uses another distributed index and a system of queries so that a node can find any peer node connected to the network. If each node indexes a set number of peer nodes, as the network grows the number of queries needed to find a node grows with it. The Kuziva network uses the philosophy that it is better to keep the number of queries low and increase the number of peers each node indexes. That number must increased in proportion to the logarithm of the number of nodes in the entire network.

The average cost per node of indexing its peer nodes is determined by the number of peer nodes it indexes divided by the length of time each node is connected to the network. For example, if a node indexes a thousand peer nodes that stay online for days, it will use roughly the same amount of bandwidth as another node that indexes a hundred peer nodes that connect and disconnect from the network regularly.

The last important cost is the cost of the actual searches. Since a decidable network keeps a sorted index of all the files, the searches do not use very much bandwidth. In a

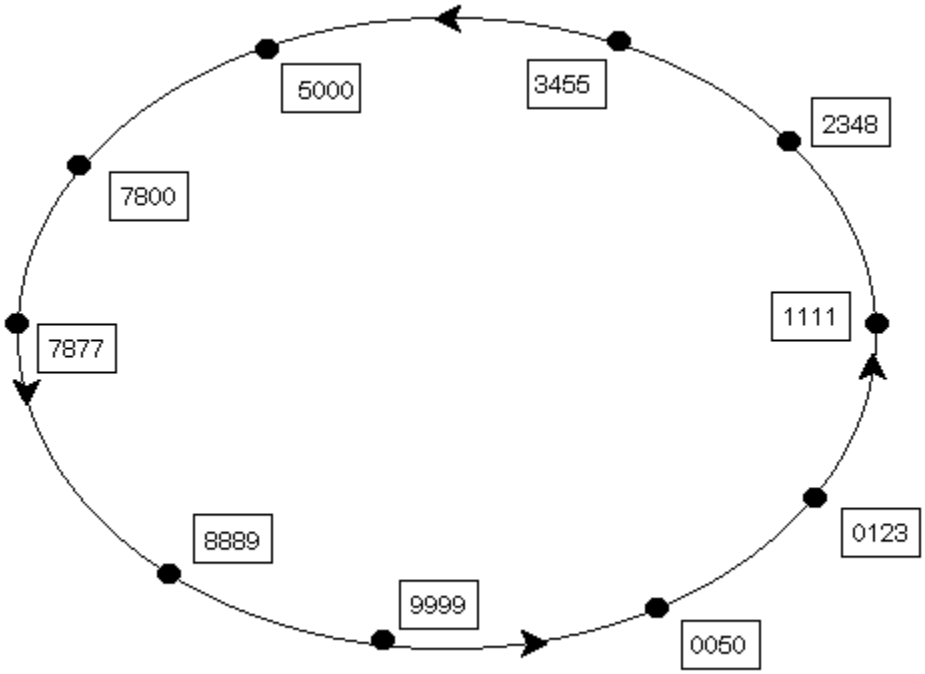
Kuziva network this cost is small compared to the cost of maintaining the index of files and the index of nodes.

Therefore we conclude a decidable network scales very well as more nodes join the network. The cost of maintaining the file index is constant, the cost of searching is constant and the cost of maintaining the node index increases logarithmically. In normal cases the savings in the cost of searching more than makes up for the cost of maintaining the indexes. The most important factors in deciding the cost are the average number of files per node and the average length of time that a node is connected to the network.

File Index

Conceptually, a Kuziva network forms a single large circle. Each node is assigned a position in the circle on the basis of its node ID. This node ID is determined from the MD5 hash of the node's IP address (an MD5 hash is a 16 byte "finger print" for a piece of data). Generally, the first four bytes of the MD5 hash are used as the node ID, but if those are already in use then the next four bytes are used instead. The nodes are ordered from lowest to highest according to the binary value of their node IDs.

Figure 1: Nodes in Kuziva network (digits represent bytes)



Each file on the network is assigned two positions in the circle between the nodes. The file's position is determined by the MD5 hash of the filename. It is assigned a position

according to the first four bytes of the filename hash and another position based on the next four bytes.

Each node is responsible for indexing the files directly after it. For example, (from Figure 1) nodes "0123" and “8889” would be responsible for a file with "0888 9862 0000 0000" as its MD5 hash because “0123” is directly before “0888” and “8888” is directly before “9862”. These nodes would have to store the filename MD5 hash, an MD5 hash of the file's content, and an IP address and port number where the file can be found.

The MD5 hash of a file's content is used to support features such as downloading from multiple sources at once, or downloading from an alternate source if one source goes offline in the middle of a download. Another feature this enables is the ability to choose a specific file from many with the same filename. For example, if an artist records a song called, "I Love Cheese" and wants people to download it, he can specify that they should download, "I Love Cheese [3e41]" where "3e41" represents the first two bytes of the file's content MD5 hash in hexadecimal. If the people downloading did not specify this then they might accidentally download a completely different file that just happened to have the same filename.

When a new node joins the Kuziva network, it first finds its own position in the circle. Next it finds the nodes responsible for indexing the files it wants to share and informs them of its IP address, port number, and the filename and content MD5 hashes for the files it wants them to index. From this point the node will periodically check to make sure that these nodes to not go offline. If they go offline the node will find other nodes to index its files. The indexing nodes also check to make sure that the node with the shared files does not go offline. If it does, then that node's files will be taken off the index.

Thus every node and every file has a defined place in the network. Two nodes index every file. The index is always kept up-to-date through the periodic checks.

Node Index

In addition to indexing files, each node in the Kuziva network is responsible for indexing other peer nodes. It keeps two lists for this purpose. One is the General Node List, which is large. The other is Nearest Node List, which just stores the 15 adjacent nodes on either side of the node itself (30 nodes altogether).

When a new node joins the network, if first determines its own ID by finding the MD5 hash of its IP address. The node then searches for its position in the Kuziva circle and informs the node directly in front of it, of its arrival. This node checks to make sure that the new node is really online and that its node ID matches its IP address. If the checks pass then the node informs the new node about the list of files it should index. The new node then sends PING queries to find all the addresses for its Nearest Node List. Each node must check to make sure the new node is online and that its ID is valid before adding the node to their Nearest Node Lists. The new node does the same checks and adds these nodes to its Nearest Node List. If a check fails at any point, the node that checked will ignore the node with the false ID.

The new node then sends a PING query to each of the 15 nodes on either side of it and they perform the same checks of its IP address and node ID. The new node also checks the MD5 hashes of the adjacent nodes in return to validate that the IP addresses and node IDs are valid pairs. All these nodes go into the Nearest Node List. Every 10 minutes the node will check to make sure that the nodes in its Nearest Node List are still online and if they are not, it will add new nodes to its Nearest Node List.

The Nearest Node List is important because nodes often redirect search queries to nearby nodes and thus it is important for nodes to know their neighboring nodes are trust worthy. Also if a node is under a high load it will ask its neighboring nodes for help.

The purpose of the General Node List is for searching (searching is explained in the next section) and helping other nodes search. It consists of a collection of addresses with matching time tokens to record when those address were last verified as online. For a Kuziva network with 1,000,000 nodes each node has a General Node List with between 5,000 and 10,000 addresses in it. The size of the General Node List is chosen so that a search should take less than 20 queries to find a file or node.

The General Node List is kept up-to-date with the UPDATE query. If a node receives a SEARCH query it sends an UPDATE query to the node nearest the thing being searched for. Otherwise nodes select random nodes from their General Node List to send UPDATE queries to. Each node sends 80 UPDATE queries per minute. The UPDATE query returns 10 node addresses with matching time tokens to show the time since the nodes were last verified as online.

Much of this part of the Kuziva design was based on estimation and will be reconsidered once more real world data becomes available. In particular, the size of the Nearest Node List, the size of the General Node List, the frequency of the UPDATE queries and the number of addresses returned in response to an update query need to be reevaluated. These things are highly dependent on the length of time nodes stay connected to the network, and on bandwidth issues.

Searching

Since the Kuziva network only stores the MD5 hash of a filename and not the filename itself, the users need to know the exact filenames (ignoring capitalization and punctuation) in order to perform searches. In a typical scenario the user would visit an artist's web page and cut and paste filenames into a Kuziva program. Besides just entering the filename the user can also enter part or all the file's content MD5 hash if they want to select a specific file from among many with the same name.

For a Kuziva program, searching for a file is a matter of finding which node is responsible to index the file and then asking that node what files with the given filename exist on the network.

To find the node responsible to index the file, the program finds the first four bytes of the filename's MD5 hash. By the design of the network, the node with the file's index will be directly before those four bytes in the circle (see Figure 1). The Kuziva program will send SEARCH queries to the node nearest to, but without going past, the four bytes.

The response to a SEARCH query will either include 5 nodes that are nearer to the node being searched for, or a response that the node being queried is the right node.

When the node responsible for indexing the filename has been found, the program will send it a query with the entire MD5 hash. The node will respond with a list of all the files available on the network with that file and their content MD5 hashes.

If the results from searching for the filename are not successful, the program can search for the backup indexes by swapping the first four bytes in the filename hash with bytes five through eight, and then repeating the search process. If the second search fails, the program rightly concludes that the file is not available anywhere on the network.

Network Security

To some extent, peer-to-peer networks are all vulnerable to malicious users because of the simple fact that peer-to-peer means entrusting data to random and often unknown users. Thus the goal of peer-to-peer file sharing protocols is not perfect reliability but reasonable reliability assuming that the great majority of users are not malicious.

However, the difference between non-decidable networks and decidable networks is that with the former it is impossible to know every node that indexes a certain file, but with decidable networks it is always possible. A malicious user on a decidable network could use this information to selectively perform denial of service attacks on nodes indexing particular files. If the attacks were successful, the files would be unavailable to anyone searching for them. The way to counter this inherent flaw is through mirroring popular files.

Besides that most important issue with decidable networks, there are many other security considerations that apply to all peer-to-peer networks. Kuziva network is based on the following security guidelines: Never allow nodes to make their own choices about correct behavior, and never allow any single node to have too much influence. In the end though, good security means analyzing each type of network transmission and questioning what would happen if each piece of information in the transmission were to be used maliciously.

To limit a node’s influence, every type of transmission needs an upper limit. For example, there are only if a node receives over a certain amount of UPDATE queries in a minute it will stop responding. Another example is that when a node is joining the network there is a limit to how many files it will start to index right away. These upper limits are needed because without them subtle denial of service attacks are possible.

One way that the Kuziva network limits nodes' choices is that it does not let them choose which or even how many files they index. It is tempting to let the nodes index as many files as the users choose but this could possibly give a malicious user too much influence on the network.

Another way that influence is limited is by sending UPDATE queries to many nodes instead of just a few. By doing this Kuziva limits the harm a node can do if it sends back incorrect replies.

In some peer-to-peer networks queries are relayed through many different nodes and this can be used to cause a denial of service attack. In the Kuziva network no queries are relayed and this limits the influence any single node can have.

In designing the Kuziva network there were many potential problems that were only noticed after evaluating each network transmission. For example, on a typical computer finding the MD5 hash of an IP address uses 100% of the CPU time for one third of a second. If a malicious user sent LOGIN queries from forged IP addresses he could easily saturate a nodes CPU and thus create a denial of service attack. The way that Kuziva corrects this is by checking that IP is correct before checking the MD5 hash. On the other hand, if the IP is correct but the MD5 hash is not, the node will be ignored after the first false LOGIN attempt.

There are many examples of potential attacks that were considered in the design of the Kuziva network, and unfortunately there is not enough space in this paper to describe them all.

Privacy

A Kuziva network stores the following information about a user: the user's IP address, a 4 byte Kuziva ID derived from the user's IP address, a port number for the user's node, a list of MD5 hashes of the filenames and a list of MD5 hashes of the contents of the files that the user is willing to share.

It is impossible for a peer-to-peer network to keep the users' IP addresses private. Nodes need to contact peer nodes and to do that they need this information. A Kuziva program does not give this information out in the user interface but a user could just use operating system features to find the IP addresses it was contacting. From a user's IP address it is possible to determine the user's identity by either contacting the user's ISP or through other channels. Thus it is impossible for a peer-to-peer network to keep the identities of its users private.

Since MD5 hashes are a one-way hash, it is impossible to determine the original filename for a given hash. The only nodes that know what the hash is (not the filename itself but just the hash of the filename) are nodes that have the original file, nodes that search for the filename and nodes that index the hash. The nodes that index a file could download it and find out its contents that way. Besides those nodes, no other nodes are able to do that

because they do not have the filename hash. The files that a node indexes are assigned at random on the basis of that node's IP address, mitigating the seriousness of this issue.

It is important to note that although with a Kuziva network it is impossible to find out all the files that a particular node is sharing, it is possible to search for a given filename and find all the IP addresses of the nodes that are sharing that file.

A final, very important matter to consider is whether searches are anonymous. The only information that a client sends over the network in a search is the MD5 hash of the filename it is looking for. This information is only sent to the one node or, in some cases, two nodes that have the index, and to nodes that have the actual file.

It would be possible to tell which nodes, for example, were downloading Brittany Spears songs by naming a text file "Brittany Spears - Oops I Did It Again" and seeing which nodes downloaded it. But other than that, searches can be considered nearly anonymous.

Thus the Kuziva network affords some level of privacy, but it is not completely private.

Load Balancing

The goal with the Kuziva network was to have each host use the same amount of bandwidth. (This is not considering the amount of bandwidth used to transfer files but just the bandwidth for maintaining the indexes and responding to search requests, etc.)

However, there are sometimes files that are extremely popular and so the nodes that index popular files are going to be over loaded. In that case the over loaded node will send a list of indexes of the popular file to the fifteen nodes directly before itself in the Kuziva circle. The fifteen nodes will start responding to search requests for that file. If some new node connects to the network and has the file, then one of these nodes will add it to their own index.

When the file becomes less popular the nodes that index it will stop sending requests for help to their neighbors. The neighbors will eventually go offline and forget that they were responsible to keep the file in their index. At that point everything returns to normal.

Specifics in Detail

The paper has discussed the Kuziva network in general terms up to this point and this section goes into more detail about certain parts of the protocol.

Joining the network

1) The new node finds its place in the network with the SEARCH query.
2) The new node verifies the ID of the node directly in front of it.
3) The new node sends the JOIN query to the node directly in front of it.

4) This front node sends a PING query to the node to make sure that it is really there and not just someone with faked IP headers.
5) The front node verifies the node's ID. If it is valid the front node adds the node to its Nearest Node List.
6) The front node sends send the new node a list of files to index.
7) The new node sends pings to the 15 nodes on either side (30 total.)
8) The 30 nodes verify that the new node is online and verify its ID and then add it to their Nearest Node List.
9) The node verifies the IDs of the 30 nodes.
10) Every 10 minutes the node will send a PING query to the nodes in its Nearest Node List to verify that they are online. If they are not online the node removes them from its Nearest Node List.

Figure 2: Types of data stored on each node

Local Information The node's Kuziva ID The local shared files A list of filename and content hashes for the shared files The local IP address and port This information is stored over multiple sessions.
File Index A list of filename hashes, with matching file contents hashes, IP addresses, and port numbers Estimated amount: 600 indices This information is not stored after the end of a session.
Nearest Node List A list of the nearest nodes that are online and their addresses Amount: 30 This information is stored over multiple sessions.
General Node List A list of nodes that have been verified as online within the last 15 minutes but are not in the Nearest Node List Estimated amount: 5,000 to 10,000 addresses for a 1,000,000-node network This information is stored over multiple sessions.

Adding a file to the index

1) The node with the file finds the filename hash and the contents hash.
2) The node does a SEARCH to find the first 4 bytes in the circle.
3) The node verifies that the ID of the node responsible for indexing its file is correct.
4) The node sends a NEWFILE query to the indexing node.

5) The indexing node sends a PING back to make sure that the sharing node is really online.
6) The indexing node verifies that the file sharing node's ID is correct.
7) The indexing node adds the file hashes to its index.
8) The sharing node swaps the first four bytes with the next four bytes and repeats step 2 through 7.

The update cycle

Each node needs to keep its General Node List up to date. The aim is to have between 5,000 and 10,000 online nodes in the list.

If the node has logged in before, it will have a list of IP addresses stored from the last session. The node can send an UPDATE query to each address in that list. After processing that list, the node can cycle through its General Node List sending an UPDATE query to each address. Eventually the node will have over 10,000 addresses in its General Node List and can slow down.

The UPDATE query works in the following way: If node "0050" in Figure 1 has nodes "5000" and "9999" in its General Host List it could send an "UPDATE 9999" to node "5000" and get a reply that had the node ID, IP address and port number of 10 nodes between "5000" and "9999." If node "5000" did not know 10 nodes between "5000" and "9999," it would respond with random nodes that it did know about.

PING and REVERSE-PING

If node "0050" in Figure 1 sends a PING query to node "0123" with a random byte of data, node "0123" would reply with same byte of data and the address of the node directly after it. In this case, that is node "1111." REVERSE-PING is the same as PING except that the reply will have the address of the node directly before instead of having the address of the node directly after. REVERSE-PING is used in the LOGIN stage to find the addresses of the 15 nodes in front of the querying node.

Searching

If a node wants to search for the file "Three blind mice.jpg" it will take a hash of the filename to get 16 bytes. It will then send a SEARCH query with the first 4 bytes from the hash to the node nearest to where the data should be. The reply to that query will either be a "yes" or a "maybe" or the addresses of 5 nodes that are nearer to where the data should be. The "yes" reply means that this is the node that should know about that data. The "maybe" means that the node responsible for that file became

overloaded and offloaded some of its indices to this node so this node may have that file but may not.

The nodes that receive a SEARCH query respond to the search and then they use the UPDATE query to learn 10 nodes that exist between the nearest node that they knew about and the thing that was searched for. (This could be changed to more than one UPDATE after testing). The next time the nodes receive the same SEARCH query they will be able to respond more accurately. One could say that the nodes learn more about the area for which they receive the most SEARCH queries. By the design of the network, they receive more SEARCH queries about the parts directly after them in the circle.

Once the searching node has found where the data should be it sends a RESULTS query with the entire filename hash and gets back a list of file contents hashes and addresses where the file can be found. If no responses are found then the searching node can swap the first four bytes of the filename hash with the second set of four bytes and repeat the search. If there are still no results found then the file is not available on the network.

Once the node has a list of addresses where the file can be found, the node will send a GETFILE query to one or more of these addresses and in response get a port and some information about the size of the file and the speed of the connection.

The node can then download the file from the given port.

OVERLOADED

Occasionally a node will become over loaded indexing and responding to queries for a popular file. If a node has more than 600 indexes for the same file, it will send an OVERLOADED query with 10 of its indexes to all the nodes before itself in its Nearest Node List. The number of indexes needed before a node is considered overloaded may change after testing.

Disconnecting

When a node leaves the network it gives its list of files to the node directly in front of it in the Kuziva circle.

Conclusion

Decidable networks offer are highly scalable in terms of average bandwidth usage per node in the network compared to the number of nodes on the network.

Kuziva is a decidable network because there are two nodes that can be queried to determine whether a file is or is not on the network. The actual nodes to query in order to find this information depends on the filename but the formula to find the nodes is simple and fast.

Kuziva network is interesting because it is a decidable network but there are some things in the design that are useful for other types of peer-to-peer networks. Kuziva uses MD5 hashes instead of filenames and this saves bandwidth and provides a level of privacy. Kuziva lets users select the exact file that they want from multiple files with the same name by specifying the first four hexadecimal values of the file's content hash. Most current peer-to-peer programs do not support this powerful tool. Lastly, the use of a node's IP address to determine and limit the node's influence on the peer-to-peer network is a good security feature.

Currently there is a prototype Kuziva network under development. It will lack most of the features of a full implementation and as it is only meant to test scalability and bandwidth usage.

Peer-to-peer protocols examined in the design of Kuziva

Gnutella: http://www.gnutelliums.com/
Freenet: http://freenet.sourceforge.net/
Napster: http://napster.com/
MojoNation: http://www.mojotainment.com/

Acknowledgements

I would like to acknowledge Dr. Marty Wolf for his encouragement, for providing valuable feedback during the design of Kuziva, and for his assistance in preparing this paper.