Allegheny College
Department of Computer Science

Senior Thesis Proposal

The Capabilities of Peer-to-Peer Technologies

by

Charles Misback

ALLEGHENY COLLEGE

COMPUTER SCIENCE

Project Supervisor: **Professor Mohan**
Co-Supervisor: **Professor Kapfhammer**

7 March 2021

Abstract

The Capabilities of Peer-to-Peer Technologies

Peer-to-Peer (P2P) networks have largely been neglected in favor of traditional Client-Server networks which typically rely on large companies to purchase and control massive server farms as hosting. In contrast, P2P networks rely on the every day person to "crowdsource" power to complete tasks that even Client-Server networks have trouble with. The neglect towards P2P networks has made it difficult to demonstrate their numerous advantages over Client-Server networks. This paper is proposing the creation of a relatively basic P2P file sharing network which implements many of the features found in the popular P2P file sharing network known as Bittorrent. Along with this, a sister client-server network will be developed with the goal of being a suitable means of comparison during testing alongside the P2P network. Some of the parts that the testing will focus on is performance, scalability, and security. Through this tool, I hope to show the reader some useful advantages of P2P networks which client-server networks lack.

Introduction

The goal of this project is to investigate Peer-to-Peer (P2P) technology and the specific advantages that it possesses over "normal" client-server architecture.

To understand the intricacies of P2P networks, it is important to understand what a decentralized network is and what this unique network architecture means for users. A centralized network is structured how you would expect considering its name, users communicate with one central server. All network activity passes through this server. To contrast, a decentralized network has no "central" server; each user communicates with at least one (usually more) other client that is currently on the network.

In order for a decentralized network to operate successfully, each node (a single user) must be of equal importance to every other node; they all possess the means to single-handedly operate the network. If a single node is shut down, the network will continue to persist. If that node happened to be the central server of a client-server network, the whole network would be shutdown. Stability like this is the main advantage of a decentralized network.

Something that goes hand in hand with stability is security. It is more difficult to target multiple nodes than one central server so malicious activities such as hacking and ddosing become less effective. This also enables a very reliable means of verification which utilizes the multitudinal nodes to ensure the validity of requests made by users.

P2P networks can be split up into three main groups: unstructured, structured, and hybrid. If a P2P network is "structured" that means there is a protocol which enables nodes to quickly search for one another, usually in the form of a distributed hash table (DHT). Unstructured simply means the lack of a protocol, each node randomly connects to other nodes. With no protocol, unstructured P2P networks struggle with connecting specific users to each other, having to search through the whole network, consequently consuming lots of processing power. However this lack of protocol also allows users to quickly connect and disconnect without having to jump through any hoops. Hybrid models combine both P2P and client-server architecture through a central server acting as a sort of air traffic controller, directing which node for a user to connect to.

Initial research shows that there are mainly two different types of P2P applications that exist currently: file sharing and computing. File Sharing is currently the most popular with BitTorrent leading the P2P charge; providing high bandwidth downloads for popular files with no server overhead. P2P Computing is a very different story. The idea of a P2P computing is to use a user's device when they're not in use, running computations to fulfill the needs of the entire network.

I believe the use of P2P technology could greatly decrease the need of large server farms. Humanity creates a lot of waste and wasting computing power is a large one. Networks that could be optimized by utilizing new, more efficient technologies continue to use outdated and increasingly wasteful software. On top of this many users will benefit from the increased download speeds which can be sought from a P2P file sharing network.

Related Work

The related work in P2P networks is not as numerous as standard networks obviously so I will choose to explore the work of Bram Cohen, the creator of bittorrent. Cohen wrote a research paper going into depth on some of the key algorithms that are integral to the success of the network. These are focused on the unique idea of splitting up a data file into numerous pieces, storing them on a user's computer once downloaded, an assortment of incentives to sustain users in a network, and methods of searching for files directly on the network. [1]

The easiest way to understand the bittorrent protocol is to start with a torrent file. A user creates a .torent file which contains information about the file, the length, name, hashing information, and the url of a tracker. A tracker is a user which directs incoming leechers (people wanting to download the file) to those which are known to have the desired file. Those wishing to download use the bittorrent protocol to request this list of possible peers from the tracker. The protocol runs on top of HTTP and contains necessary information such as useful networking information to link peers together. From here, downloaders send diagnostics about the file which they have and which part. SHA hashes are supplied to the user via the .torrent file to ensure the integrity of the incoming data file which prevents seeders (peers which share the data) from spoofing any downloads for the user, possibly with some kind of malicious intent.

Sometimes this does not work as intended, as the file itself can already be malware infected. One solution to this is creating a percent satisfaction that is raised by positive feedback and lowered by negative feedback. [2] Seeders send data in quarter megabyte pieces which enables a user to not become independent on just one user and instead spreading out the download among all available peers. This continues while a file is being downloaded by a user.

Some optimizations which help bittorrent have such a great repuation for download speeds is the process of breaking pieces of a file down further down into sub pieces, roughly 16 Kb in size, and always having at least five requests queued at once. The size of the queue has been calculated based on an average value which saturates most connections and uses the network in the most efficient way possible. A user will also prioritize requesting these rare sub-pieces over more common ones in order to maximize the chance of downloading the entire file since a seeder could terminate at any point. [3] In addition to this, the very first piece which is sent to a user should be quick to download in an attempt for the downloader to get a whole piece as quick as possible so they can start seeding that piece to other peers. Though as soon as this is done, rarest first is resumed. As a download is close to finishing, endgame mode is triggered. This forces a downloader to request all the sub-pieces of the piece that they're currently on at once from all peers, which may seem to be redundant but makes sure that a download doesn't hang at the very end. A measure such as this one may seem wasteful since some requests will already be fulfilled when a peer gets around to it but the amount of bandwidth wasted is negligible.

This brings us to one of the most important aspects of maintaining a P2P network's longevity, incentives. When a user stops seeding a download to a peer but still accepts downloads from it, this is called choking. This may seem backwards, but doing so always strives for the highest possible download/upload speed through taking advantage of TCP's built-in congestion control and stopping freeloaders (users which refuse to upload). [4] Choking is a tit-for-tat policy which is based on the idea that a user should be able to request as much data as they have downloaded for themselves. If a peer's upload speed to you is the lowest out of all other peers, it is choked. The issue is created when a user can only have a certain amount of unchoked peers (default is four), so in order to maximize download speed the user must search for peers which upload to the user the quickest. One glaringly large issue with tit-for-tat is the restriction of the network, punishing peers with low bandwidth even if it was only for a short time. To fix this, bittorrent clients use optimistic unchoking to unchoke a random peer no matter its performacne once every 30 seconds. For reference, tit-for-tat choking and unchoking occurs every 10 seconds. This period of 10 seconds is long enough for a peers upload to ramp up to full capacity so as to not bog down users with constant choking and unchoking. [5]

A very interesting strategy that some bittorrent client implement is anti-snubbing, caused when a user hasn't received data from a peer for a minute. The user will then refuse to upload to that peer until it is optimistically unchoked, which consequently forces the user to find other peers and not waste time seeding data to a peer which won't recipcrocate. [6] This only contintues while no data has been uploaded so if that previously ignored peer optimistically uncokes the user, the tit-for-tat relationship is restored. This feature is typically only used when lots of other peers are available to prevent total snubbing. Overall, bittorrent is a very promising and extremely interesting network and in many ways, a large experiment for larger, more widespread and mainstream P2P networks. It is an example of a P2P network without any central server whatsoever, except for distribution of the .torrent file. [1]

Tit-for-tat is not the only way to incentivize peers to upload to the network. There is lots of research on this topic. This includes round-robin based selection of peers, an auction based system [7], the clusetering of similar-bandwidth peers [8], and a new proposed method to directly improve upon tit-for-tat [9]. When developing systems such as these, it is prudent to attempt to break them; simulate exactly what people may attempt in the real world. [10] Doing so finds weaknesses and strengths of methods. The above comments on some of the avenues that bittorrent takes to attempt to reduce this.

Mentioned briefly in the introduction, a Distributed Hash Table (DHT) is a replacement for the aforementioned .torrent file. A DHT is a means for peers to index the current files which they are willing to upload and share this knowledge among the P2P network. A peer can send a search query for a file which they or their directly associated peers may not have, and the query will propogate throughout the whole network. (Alg. 1) When a peer confirms that they have the requested file, they start to upload it to the original peer which requested it. There are varying methods of implementation for this but the most well known is using a Kademlia DHT. [11] A particulary interesting solution to this is cubit, a scalable and effiencet P2P system which finds files on other peer's systems closely similar to a search query. [12]

Algorithm - 1 Handle Query

Input - reqPeerid, query, hops
Output - None

```
1:  for (fname in self.files.keys()) do
2:     if (query in fname) then
3:        self.connectandsend(peerid, fname)
4:        return
5:     end if
6:  end for
7:  if (hops <= 4) then
8:     for (peer in self.peers.keys()) do
9:        self.connectandsend(peer, reqPeerid,
          query)
10:    end for
11: end if
```

Though this is not the main focus of the proposed project, there are many other interesting aspects of P2P networks such as crowdsourcing compute power in order to process millions and even billions of different protein folding patterns to find a suitable one for attacking a virus. Qmachine was developed to bring this ability to modern day web browsers, stressing the importance of large cross platform mobility to reach as many people as possible. The authors found that the average amount of compute power which comes from the daily traffic of a high traffic website is well within the HPC (High Performance Computing) range, so if implemented into such a website could supply super computer level performance on demand. [13] It's important to mention that this is most certainly not an unstructured P2P network so it does not come with all the privacy capabilities of unstrucutred P2P networks such as bittorrent which everyone is

a stranger to each other.

Method of Approach

The goal of the project that will be proposed is simple, but reaching it contains many obstacles. It is to convey a deep understanding of the current P2P technologies and create a working P2P network which showcases current P2P optimizations such as those discussed in the bittorrent network. This project will be ongoing and experimental, so the goals and specific methods of reaching them may change depending on viability of said network.

The first concrete goal is to create a python network which utilizes the native python networking api to construct a virtual p2p file sharing network. The approach to programming will be object oriented to make a very intuitive and easy to understand simulation. While this may not be the most efficient method performance wise, it seems like a good start to introduce readers to the practicality of p2p networks. Tests can then be run on this to find the speed and distribution ability of the little optimizaitons. These tests will try to be as relfective of a real network as much as possible, closely adhering to the principal of awareness between peers. A peer should only be aware of its neighboring peers. That means all information must be passed through the network connection between the two which in this case will be a python network connection.

The feasibility of this goal is high because I already have created an operational P2P file sharing application which has the ability to connect to multiple peers, propogate messages throughout the network to search for files, a communication protocol, and file downloading capabilities. None of the features mentioned below have been implemented as of yet. The network was created with the help of an open-source P2P network.

Algorithm - 2 Send File Chunk

Input - peerConnection, fname, startIndex, size
Output - None

```
1: if (fname not in filelist) then
2:     return
3: end if
4: file = open(fname, r+b)
5: file.seek(startIndex)
6: r = file.read(size)
7: peerConnection.senddata(r)
```

The second concrete goal is to explore different strategies of splitting up files into smaller parts (Alg. 2)which is vital to increase the performance of a p2p file sharing network and hashing each of these chunks to verify their integrity between file transfers. The approach which bittorent takes is the creation a hashtable which a user downloads in order to assemble the numerous pieces and sub-pieces (Alg. 3)

Algorithm - 3 Receive File Chunk

Input - fname, startIndex, size, data
Output - None

```
1: if (fname not in torrentList and not
   verify_chunk(Input)) then
2:     return
3: end if
4: file = open(fname, r+b)
5: file.seek(startIndex)
6: file.write(data)
7: file.close()
```

as well as ensuring integrity of each piece using SHA-256. (Alg. 4) Once this is implemented, comprehensive testing regarding download speeds can be conducted. The file chunking will enable extremely fast download speeds for our P2P network.

Algorithm - 4 Verify File Chunk

Input - fname, startIndex, size, data
Output - True or False

```
1: chunk = (startIndex, size)
2: torrent = torrentlist[fname]
3: if (chunk in torrent.chunks) then
4:     hash = haslib.sha256(data).hexdigest()
5:     if (torrent.chunks[chunk]) == hash)
       then
6:         return True
7:     end if
8: end if
9: return False
```

The last concrete goal is to implement the other advanced algorithms which bittorrent utilizes and were described in the related work section. To aid with piece selection there is the Random First Piece (RFP) algorithm, the Rarest First algorithm, and Endgame Mode. Piece selection alone does maximize download rate; this is done by a choking algorithm. It operates under a tit-for-tat policy while utilizing an occasional charitable algorithm to give new peers a chance.

A final, moonshot goal would be to implement the Kademlia DHT algorithm as discussed before. This is a very complicated algorithm but it is currently a very popular organizational method for P2P networks. The reason that other goals are prioritized over this is because a DHT does not affect the download or upload speed of files. It only increases discoverablity and longevity of the network.

Multiple goals are necessary to ensure the completion of the proposed project and will future proof the project in the case that one goal is easier than anticipated to meet. It allows pacing to be judged based on experience during the project as opposed to setting it before and potentially crashing and burning.

Evaluation Strategy

To go into detail regarding testing, the exact goal of the test will be stated. The goal is to find desirable balances between certain aspects of the application such as the success of choking peers, general performance, performance under stress, longevity, and best/worse case scenarios. The results of this test will greatly impact the direction of this project. For the time being this is a proposal which is seeking to find benefits that p2p networks have over traditional networks, but if a different result is found the project should still be continued but with a different attitude and direction. A strict comparison between a traditional network would have to be created, so it would make sense to make a similar python environment which simulates typical traffic on a website and test it head to head with the p2p network.

This was mentioned briefly in the method of approach section, but a good way to evaluate the success of the p2p simulated network is to build a typical client-server network. Doing this will allow the comparison of various important traits which users value in their networks. Some of these are performance, reliability, longevity, ease of use, security, among many others. Obviously if some of the features fail to be implemented into the P2P network, those will be left out since it would be unfair to compare the two based on features that are not implemented. The creation of the client server simulation will be purely a tool for testing, so it may be useful to create it while developing the P2P simulation so as to aid the development.

Some of the features that will be implemented for sure are the basic file transfer mechanism, choking/unchoking, security, and ease of use. The file transfer for mechanism for the typical client-server network will most likely be ftp with the P2P mechanism being the splitting of a data file into pieces and creating a hash table corresponding with those pieces allowing files to be rebuilt from little pieces.

Table 1: Data sets for file sharing

Data File Type	synthetic?	Sizes: KB	MB	GB
Text File	yes	1, 10, 100,500	1, 10, 100, 500	1, 5, 10
Video File	no	n/a[a]	10, 100, 500	1, 5, 10
Audio File	no	n/a[a]	1, 10, 100, 500	1[b]
Zip File	yes	1, 10, 100, 500	1, 10, 100, 500	1, 5, 10

[a] Smaller sizes for these file types would not make sense
[b] Larger sizes for these file types would not make sense

Choking/unchoking will be compared to other current day networks such as bittorrent to check if it is being done optimally. Bittorents explanation of choking/unchoking is very clear so a test like this should be fairly straightforward and can most likely be done during the creation of the P2P network simulation. Security is more complicated but implementing a modern form encryption will work out just fine, especially for a simulation. If the project were to get into the stage of real world testing, encryption similar to the tier which bittorrent implements would be vital, since anyone could gain access to the network. Ease of access will be tested through the interaction with another user to attempt to send a file or another interaction between them which ends up being implemented. Feedback on this topic will be very valuable so again, this is going to be a recurring thing throughout development.

Going into more detail, Table 1 lists testing file transferring on both networks with assorted files of many different sizes. The purpose of this is to test the capabilities of the network and see if there is a lower or upper bound to the file sizes. The dataset which I intend to use for testing can be divided into two main categories, binary and non binary files. An example of a binary file is a photo, video, executable, anything that can't be read by a simple text editor. A non-binary file is one which has a well known character encoding which is typically utf-8. The final dataset will contain both of these file types with varying sizes as seen in Table 1.

To test the P2P against the Client-Server nework the conditions need to be equivalent for both of them. For the P2P network these conditions are number of peers, max number of neighbors for each peer, and number of connected peers. The equivalent conditions for a Client-Server network is number of clients. From here, a file from one of the datasets will be given to the central server for Client-Server and a variable amount of peers for the P2P. The given task is for a variable amount of peers or clients to download the file. The time taken to complete the task will be what is measured as well as the progress made. By varying the variables to which I mentioned lots of valuable data can be ascertained.

Based on these results, the decision will not be binary. Each network has their areas and conditions that will make them undisputably the best. All that this project will be trying to prove is that the existence of P2P networks should not be ignored because there are many benefits which client-server networks do not have.

Conclusion

In conclusion, this paper is proposing the creation of a P2P network programmed in python which will help demonstrate the capabalities and benefits which it has over a traditional client-server network. This will be done using another similar tool which a client-server network instead of P2P. A comparison between

the two will occur where each of them is tested on certain aspects which are both unique and common to both of them.

The impacts of this project won't be anything special. Nothing new is really being proposed, it is just attempting to bring P2P to the light where it deserves to be. P2P networks used to be the only means of accessing the internet, back before the internet was widely accessible. Now, client-server networks dominate the modern internet, forcing users to use a website which is the only access to a certain type of content. This is great for companies to make money from users but it stands in front of progress. Giving everyone the chance to contribute to a network through simply downloading an application where they recieve something in return for sharing their computer with the network would give many people opportunities. Opportunities not available under an internet monopolized by large corporations whose sole interest is making money.

The potential directions for future work based off the results of this proposal are innumerable. Even if a result which ends up showing that P2P networks have negative qualities which make them worse than client-server networks, that doesn't ignore the fact that there have been many successfull P2P networks in the past, some of which are still operating today. A result like that just means the specific implementation was not better than another specific implementation of a client-server network. Results which are the opposite of that could possibly spark others to interact more with P2P networks. Even just a simple interaction with a network does more than one might think. Not only are they benefiting from the network, they are adding themselves to the network for that short time. Contrast that with a normal client-server network, most of the time only the website owner benefits from a user visiting. Instead of paying for a website through ads you could just share a small

Table 2: Research Schedule

From	To	Item
Present	November	Finish Thesis Ch 01 Redisgn Prototype UI Research specifics of P2P Implementations
November	December	Finish Thesis Ch 02 Implement advanced search querying Implement file chunking and hashing
January	February	Implement peer incentive algorithms Finish Thesis Ch 03
February	March	Finish Thesis Ch 03 & 04 Implement DHT[a] Begin testing Implement peer incentive algorithms Write majority of Thesis
March	April	Finish up Testing Finish up Writing Peer Review

[a] Will determine later if doing so is necessary, depending on current progress in project

amount of computer power which you are not using at the moment.

Bibliography

[1] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, pp. 68–72, Berkeley, CA, USA, 2003.

[2] F. R. Santos, W. L. da Costa Cordeiro, L. P. Gaspary, and M. P. Barcellos, "Funnel: Choking polluters in bittorrent file sharing communities," *IEEE Transactions on Network and Service Management*, vol. 8, no. 4, pp. 310–321, 2011.

[3] J. A. Johnsen, L. E. Karlsen, and S. S. Birkeland, "Peer-to-peer networking with bittorrent," *Department of Telematics, NTNU*, 2005.

[4] T. Locher, P. Moore, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," 2006.

[5] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent," in *Proc. of NSDI*, vol. 7, p. 4, 2007.

[6] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, "Influences on cooperation in bittorrent communities," in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pp. 111–115, 2005.

[7] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pp. 243–254, 2008.

[8] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 301–312, 2007.

[9] W. Liu, D. Peng, C. Lin, Z. Chen, and J. Song, "Enhancing tit-for-tat for incentive in bittorrent networks," *Peer-to-peer networking and applications*, vol. 3, no. 1, pp. 27–35, 2010.

[10] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting bittorrent for fun (but not profit)," in *Proc. of IPTPS*, 2006.

[11] A. Grunthal, "Efficient indexing of the bittorrent distributed hash table," *arXiv preprint arXiv:1009.3681*, 2010.

[12] B. Wong, A. Slivkins, and E. G. Sirer, "Approximate matching for peer-to-peer overlays with cubit," tech. rep., 2008.

[13] S. R. Wilkinson and J. S. Almeida, "Qmachine: commodity supercomputing in web browsers," *BMC bioinformatics*, vol. 15, no. 1, pp. 1–12, 2014.