

Peer-to-peer cached data distribution network as an alternative for the client-server model

**N.H.L.C.Samarasinghe
2019**



Peer-to-peer cached data distribution network as an alternative for the client-server model

Degree of Master of Science in Computer Science

**N.H.L.C.Samarasinghe
University of Colombo School of Computing
2019**



Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: N.H.L.C.Samarasinghe

Registration Number: 2016MCS099

Index Number: 16440998

Signature:

Date:

This is to certify that this thesis is based on the work of Mr. N.H.L.C.Samarasinghe under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:
Supervisor Name:

Signature:

Date:

Abstract

For over three decades, the client server architecture has dominated the software development industry as the go-to solution of building a web based solution. Even with the drawbacks such that it possess such as, minimum bandwidth from client to server as the maximum bandwidth and single source of failure, the client server model is still considered the most viable solution due to its simplistic approach and relative ease of scaling.

The main aim of this dissertation is to provide a new architectural strategy to compete with the client server architecture based on the peer to peer data distribution model proposed by the BitTorrent algorithm in conjunction with the browser based peer connection protocol of WebRTC. This study is focused on proposing a new architecture to overcome the limitation of client server architecture while incorporating modifications to the basic BitTorrent algorithm to provide a more suitable approach to finding peers in the fast shifting environment of websites. This thesis will discuss on how the BitTorrent algorithm's peer selection can be altered by introduction of return response time for better peer identification and how to maintain fairness for all parties in this environment.

This research looks into how the performance of the web solution can be improved with minimal additional external components while maintaining standard protocols to provide a cache based data distribution as an alternative for the client server architecture. In this research the positive impact of adding the return response time for better selection of peers and the fairness calculations considering the upload and download, to remove the biased peer selection in BitTorrent will be taken into consideration comparing to the improvement of speed and download rates with respect to the client server architecture.

Acknowledgements

I would first like to thank my thesis advisor Dr Ajantha Atukorale at UCSC as the mentor who guided me in this research through every difficulty that I faced. I also would like to thank the interim evaluation panel who directed me in the correct direction of this research. My thanks also extend to the University of Colombo and the UCSC who provided me with necessary infrastructure to allow me to continue my work to the best of my potential.

Table of content

Declaration	iii
Abstract	i
Acknowledgements.....	ii
Table of content	iii
List of figures.....	v
List of tables	v
1. Introduction	1
1.1 Background.....	1
1.2 Problem statement	2
1.3 Objective of study	3
1.4 Scope of the study.....	5
1.5 Limitations	6
1.6 Structure of thesis	6
2. Literature Review.....	7
2.1 Client Server architecture.....	7
2.2 Torrent based peer-to-peer connections via BitTorrent	9
2.3 WebRTC for peer-to-peer communication	12
2.4 Fusion of BitTorrent and WebRTC	14
2.5 Conclusion.....	16
3. Research Methodology.....	17
3.1 Introduction.....	17
3.2 Research Strategy.....	17
3.3 Research Approach	17
3.4 Research Design.....	18
3.4.1 Connection establishment	18
3.4.2 System scaling	21
3.4.3 Data Gathering.....	22
3.5 Evaluation Plan.....	23
4. Proposed Solution Details	25
5. Evaluation and Results	28
5.1 Concurrent connection establishment	28
5.2 Data acquisition with multiple peers	32
5.3 Determining fairness for peers.....	35
6. Conclusion.....	37
6.1 Future work.....	38

References	39
Appendix	41

List of figures

Figure 1 - Client server vs Peer to peer communication.....	3
Figure 2 - Comparison of the advantages of client-server over peer-to-peer communication[3]	8
Figure 3 - Minimum bandwidth governing the speed of connection[5]	9
Figure 4 - Basic BitTorrent architecture	10
Figure 5 - WebRTC handshake between two clients.....	12
Figure 6 - Simplified architecture for WebRTC peer connection	13
Figure 7 - Virtualized client implementation	19
Figure 8 - Two client data upload/download rates	20
Figure 9 - Two client data upload/download rates	20
Figure 10 - Initialized docker virtual hosts	21
Figure 11 - Server and uploaded file.....	22
Figure 12 - Virtual Firefox client fetching the data from the server.....	22
Figure 13 - RTT from client to seeder	26
Figure 14 - Four clients RTT example.....	26
Figure 15 - Peer heartbeat	27
Figure 16 - Server data transfer activity.....	29
Figure 17 - Client data transfer activity of Client-Server model.....	29
Figure 18 - Client data transfer activity of P2P model	29
Figure 19 - Time taken for parallel download.....	31
Figure 20 - Download rate vs number of concurrent peers.....	31
Figure 21 - Time taken with completed clients	33
Figure 22 - Download rate vs completed clients	34
Figure 23 - Downloading data from multiple peers.....	34
Figure 24 - Choking behaviour.....	35
Figure 25 - Unchoking behaviour.....	36

List of tables

Table 1 - Time taken for parallel download	30
Table 2 - Time taken with completed downloads.....	32

1. Introduction

1.1 Background

With the rapid growth of the internet, the number of websites, users and internet based applications have grown exponentially in the last two decades. This increase in demand has caused the web hosting parties to pursue new approaches in the domains of hosting, scaling and maintenance. Even with the advancement of technology, such as increased computational power, bandwidth and storage, it still is a considerable effort to provide supply to this ever increasing demand.

Multibillion dollar hosting companies have come up with different architecture styles and strategies to meet these demands to ensure that the users are always treated to the best service. Services such as Google cloud platform, Amazon Web Services and Microsoft Azure are at the forefront by providing various services that ensure the ease of deployment and maintenance to the customers. Even though hosting strategies which provided infrastructure as a service to host the applications such as RackSpace have declined due to a more cost effective, reliable and extremely customizable approach of platform as a service model, some applications still use the model due to various reasons.

With the establishment of various hosting strategies, application backend are in the process of moving away from monolithic structures to a fine grain micro service architecture but even though research is being carried out and different approaches such as ‘Project Mosaic’[1] has been proposed, there is no evident strategy to move away from monolithic approach to the application front ends at an enterprise wide scale at the moment.

Due to this bottleneck, to provide faster content access to the consumers, different approaches have been implemented to optimize different areas of the application. Various architectural approaches such as server scaling, introduction of content delivery networks (CDN), data caching and compression has been developed to avoid these bottlenecks at higher cost and computational costs.

Although this scaling approach may be suitable in some instances, it may not be the most cost effective approach to the problem at hand. Better environment structuring, optimized placement of data and data caching can also be used in order to improve the server response times and to reduce the strain on the servers. One of the extensions to the optimized placement of data can be seen in peer-to-peer networking approaches which still only hold a small

foothold in the current hosting domains whose major approach is the client-server architecture.

With the vast improvements in the peer-to-peer networking domain we have seen many avenues where it can be adopted instead of the existing client-server architecture. With the help of upcoming technologies such as Web Real Time Communication (WebRTC) working with massively scalable architectures proposed by BitTorrent, it is possible to distribute the server workload among a distributed ad-hoc peer network. During this thesis, we will look at how such architecture with optimizations can be utilized in enterprise level applications.

1.2 Problem statement

Server scaling can be a complicated and a costly process when trying to improve the performance and the availability of the application. Introduction of costs associated with upgrading existing servers or introducing new servers as well as added complications associated with load balancing and maintenance will play a main role in these deployment strategies. These complications can be mitigated by distributing the workload among the participants similar to that of a distributed system. This architecture should support its implementation over existing protocols such as HTTP/HTTPS while being able to manage ad-hoc client base to distribute the data. One potential candidates which can support as the base for this architecture can be identified as WebRTC technology with the peer management strategies which can be seen in BitTorrent architecture.

Once the initial connections are established, we can see that the peer selection algorithm of classic BitTorrent is solely based on the amount of data uploaded by the peer. This algorithm is biased toward the selected peer who can cause the peer to be overloaded and choked in the long run. So it is needed to dynamically select the optimal peers and take into considerations aspects such as peer latency, bandwidth and introduce fairness as improvements to the BitTorrent peer selection algorithm.

By the illustrations in Figure 1, we can identify how the two different architectures behave when creating the connections and transmitting the data.

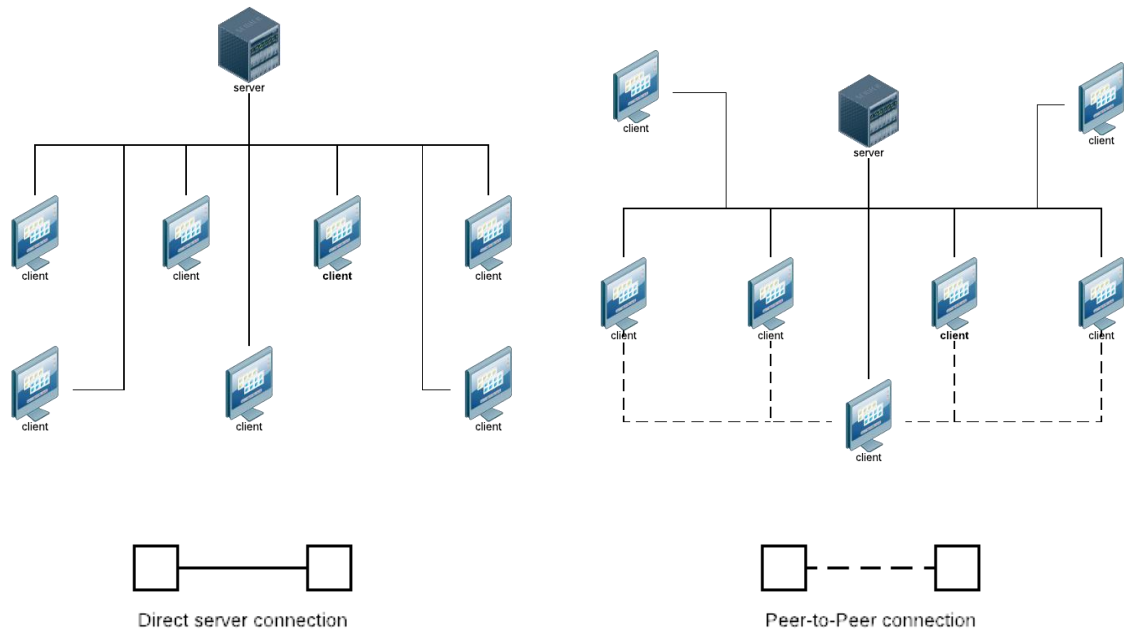


Figure 1 - Client server vs Peer to peer communication

1.3 Objective of study

The main objective of the study is to propose a new architecture as an alternative to the client server model using the peer-to-peer ad hoc network with data caching to improve the performance and to reduce the load on the server.

To accomplish this task, peer connection establishment and communication will be conducted over HTTP/HTTP protocols with the aid of WebRTC technology, while utilizing the ad hoc peer selection and scaling techniques that is observed in BitTorrent architecture implementations.

There are four major questions that this research is trying to address.

1. Determine the performance improvements comparing the proposed architecture against the generic client-server model.

With regards to the generic client-server model it is possible to see an improvement in server performance considering the processing and bandwidth usage in the new architecture. It is important to determine the improvement against the number of connections.

2. Minimize the overhead of establishing the peer-to-peer connections

Even though it is possible to establish considerable number of peer connections, establishing a peer connection can be costly. Therefore it is important to minimize the number of unnecessary connections while optimizing the usage of the generated connection.

3. Improving the existing peer selection algorithm of BitTorrent architecture such that it is fair for all the users.

The BitTorrent architecture's proposed algorithm for peer selection depends mainly on the uploaded data size of the peer to select the best connection. This inversely affects the peer due to this biased approach. To improve the performance of the download and to have a fair usage policy for the peers, the factors such as latency, peer bandwidth should also be considered other than the uploaded data size and the uptime of the peers.

4. Ensuring that all the peers are treated fairly.

Due to the implementation strategy used in the BitTorrent algorithm, it is biased in selecting peers considering the amount of data uploaded by them. This has an adverse effect of overburdening the selected peer. The selected peer will be forced to upload the data consistently and its bandwidth will be utilized in an unproductive way for the peer. When considering a web based solution, this behaviour should be mitigated so that all the peers are treated equally.

1.4 Scope of the study

The main focus of this research is towards the implementation of the new architecture and improving the peer selection algorithm proposed in the BitTorrent algorithm, where the server load and latency reduction of the server will be in concern.

Due to the complex architecture, the scaling of the application will be done with the help of Docker technology, where many of the complications can be overlooked such as complicated network setup, security concerns and hardware and software requirements.

The implementation will not be emphasizing on areas such as managing security, data compression and lower level networking and packet handling when trying to improve the server performance and peer selection components as it leverages on the existing established protocols and techniques to accommodate the basic infrastructure.

The proposed architecture is only intended to handle HTTP GET requests and other static content as dynamic responses generated at the server is unique to each client thus it cannot be cached and distributed among multiple clients. The focus will fall more on static HTML, embedded content and resources such as images and videos. Large static data providing websites similar to YouTube and Printrest will benefit mostly from this type of solution.

In this scope, optimizing the peer selection algorithm which will include the external factors other than the uploaded content size will be considered. Areas such as peer latency will be incorporated to determine the optimal peer cluster to seed the data while maintaining the fairness of data seeding and downloading for all clients. The fairness is ensured by considering the amount of data that has been uploaded or downloaded by each peer.

As the WebRTC protocol is still under the process of being standardized the browser support will be only be provided for the Google Chrome, Mozilla Firefox and Opera browsers, within the scope of the project the support will only be provided for clients who adheres to all the requirements of the given technologies and no guarantee for mobile based clients will be provided. For the scope of this project the selected browser will be Mozilla Firefox 61.0.1 and above.

1.5 Limitations

This thesis is mainly concerned about the performance improvement that can be gained via introducing the peer to peer cache based data distribution as an alternative methodology of transmitting the data from the client to the server. For this implementation, we have considered the use of WebRTC as the suitable technology to implement the communication between the peers for the purpose of sharing the cached information.

WebRTC or Web real-time computing is still in an evolutionary state and has not been standardized to be supported on all the browsers and devices even though it was introduced in mid 2012. This means that not all the users will be able to reap the advantages that are associated with the peer to peer communications to improve the efficiency of the data transfer which is discussed in this research. But most modern browsers such as Google chrome, Mozilla Firefox and Opera have already begun the support for WebRTC as a communication protocol for some time at the time of the writing.

1.6 Structure of thesis

This thesis is broken down into six major sections, namely the introduction, literature review, research methodology, proposed solution, evaluation and results and the conclusion.

The introduction and the literature review sections will be discussing on the overview of the research that was carried out and the other related research and implementations on the same domain.

Research methodology section is focused on providing in depth understanding on the approach towards building a working model from the findings based on other research and related implementations. This section will provide references on how the internals work in the technologies that are used to produce the system for the research.

Proposed solution, evaluation and results and conclusion section will provide details on how the research was implemented and the results and the findings that were achieved by introducing the new peer to peer architecture for the data distribution. These sections will provide potential usages and extensions for the research and the overall discussion on the feasibility and adaptability of the solution with potential drawback and limitations.

2. Literature Review

2.1 Client Server architecture

The inspiration for the implementation of the internet, which was initialized as the ARPANET, was the requirement to transfer data between computers which were physically apart. With the research advancements of the packet switching network that was developed for the primitive internet, well known protocols such as the Internet Protocol (IP) and Transmission Control Protocol (TCP) were introduced to the world by Defence Advanced Research Project Agency (DARPA)[1].

The initial investigations of the development of a network of heterogeneous networks, computers and transmission media has led to the development of fault tolerant, reliable architectures which made use of stateless datagrams[1] to transfer the information to and from the recipients. More advanced algorithms such as TCP have made the transfer of the data much more reliable and streamlined even with packet loss[1], with the implementation of more efficient packet retransmission mechanisms.

As an effort of trying to transmit the data from the source location to the destination by utilizing the numerous available approaches, we can observe a multitude of designs and architecture patterns that were used in the networking domain. Some of the most prominent architecture throughout the different phases of the internet that can be considered under this category can be noted as follows[2],

- Master slave architecture
- Client server architecture
- Peer-to-Peer (P2P) architecture
- Micro service architecture
- Cloud architecture

These connections can be categorized into two main areas. Mainly, single source communication and multi-source communication. With decentralized server setups such as cloud, micro service and service oriented architecture can be confusing to be categorized into these two broad areas we can determine that other than true peer-to-peer connections, the other architectures always communicate with a single host at any given point in time.

Out of these observed categories, some have become obsolete, such as master slave architecture, due to the rapid improvement and the cost reduction on the front of personal computers.

We can observe that single source communication such as client server connections are being preferred to implement communication over HTTP/HTTPS on port 80/8080. Whereas multi source communication can be predominantly seen in torrent based file transfer implementations typically running on other ports.

It is possible to determine that many of the web based applications in the present prefer the single source communication[3] over the multi-source communication model due to several advantages such as, scalability, simplicity and evolvability[3], even though many more advantages can be seen in the peer-to-peer architecture models. This may be largely due to the control and simplicity of implementation at the expense of the hosting party.

Style	Derivation	Net Perform.	UP Perform.	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customiz.	Configur.	Reusability	Visibility	Portability	Reliability
CS					+	+	+							
LS		-			+		+				+		+	
LCS	CS+LS				++	+	++				+		+	
CSS	CS	-			++	+	+					+		+
CSSS	CSS+S	-	+	+	++	+	+					+		+
LCSSS	LCS+CSSS	-	±	+	+++	++	++				+	+	+	+
RS	CS			+	-	+	+					-		
RDA	CS			+	-	-						+		-

Table 3-3. Evaluation of Hierarchical Styles for Network-based Hypermedia

Style	Derivation	Net Perform.	UP Perform.	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customiz.	Configur.	Reusability	Visibility	Portability	Reliability
EBI				+	--	+	+	+		+	+	-		-
C2	EBI+LCS		-	+		+	++	+		+	++	±	+	±
DO	CS+CS	-		+			+	+		+	+	-		-
BDO	DO+LCS	-	-				++	+		+	++	-	+	-

Table 3-5. Evaluation of Peer-to-Peer Styles for Network-based Hypermedia

Figure 2 - Comparison of the advantages of client-server over peer-to-peer communication[3]

The main disadvantage of the basic client server model compared to the peer-to-peer architecture can be distinguished as having a single point of failure. But this has been overcome using the decentralized stateless server approach still keeping to the basic fundamentals of the generic client server approach at higher cost induction towards the hosting party due to the introduction of redundant servers, databases and load balances to the implementation. For higher availability of network resources, higher cost solutions such as content delivery networks have been used[4]. Comparisons between other aspects of the client server and the peer to peer architecture can be seen in Figure 2.

Even though most disadvantages seen in the client-server model can be overcome using the above given methods, one of the overlooked criteria that is not addressed in the setup is the bandwidth bottleneck from the server to the client[5]. Due to the heterogeneous nature of the internet, the minimum bandwidth of the channel from the server to the client will govern the maximum speed of transfer[5] as depicted in figure 3.

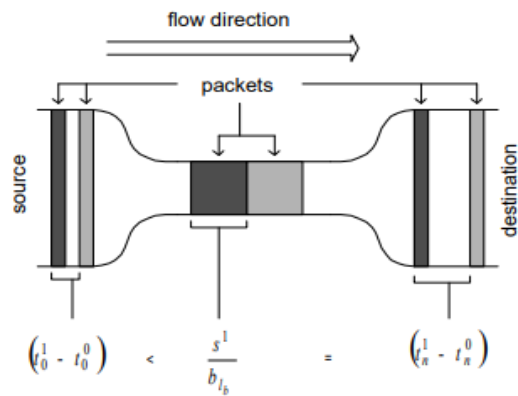


Figure 3 - Minimum bandwidth governing the speed of connection[5]

This problem can be overcome by the approach of peer-to-peer architecture implementations which can be largely seen in torrent based data sharing applications such as μ Torrent, Blizzard client and etc. These clients utilize the bandwidth to a maximum by combining a set of lower speed connections to acquire data. This ensures that the workload is shared among a group of peers which will reduce the risk of network congestion which can be seen in client server architecture implementations.

2.2 Torrent based peer-to-peer connections via BitTorrent

As discussed, the most prominent usage of peer-to-peer networking can be observed in the torrent based file sharing infrastructure. One of the most frequently utilized peer-to-peer connection algorithms can be identified as the BitTorrent algorithm, largely due to its simplicity of implementation and the approach[7]. BitTorrent algorithm has overcome some of the major disadvantages such as peer evolution, scalability, file sharing efficiency and incentives to prevent free-riding which is noticed in the peer-to-peer distributed network architecture[8] while also managing the ad-hoc behaviours in the network where “the peers rarely connect to more than few hours, and frequently for only a few minutes”[9].

The main components of the BitTorrent network can be broken down into two elements, the peers and the tracker whose relationships can be represented as in Figure 4.

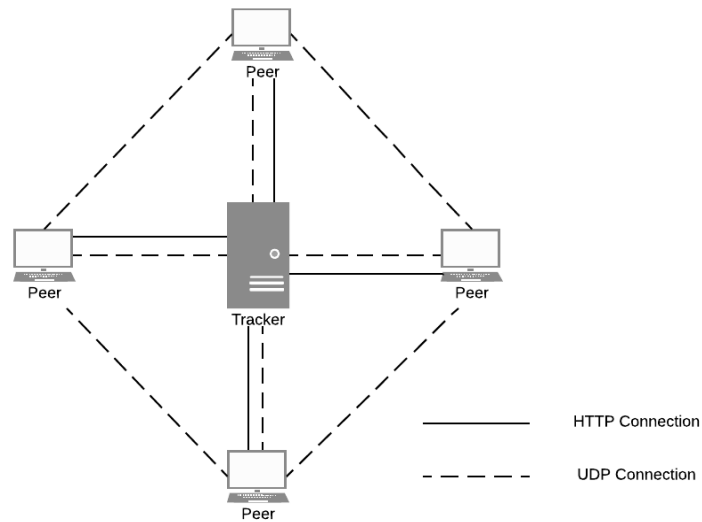


Figure 4 - Basic BitTorrent architecture

Initially the file that needs to be shared is uploaded into a machine and a metadata file known as a torrent file is generated for the file which consists of the tracker's IP, the file size, file name and its hashing information[7]. The interested parties who are going to download the file will initially acquire the torrent file and query the tracker for the information of the available peers. The file will then be fragmented and downloaded from the peers. Peers are categorized into two parts in this scenario, namely seeders, who has the complete file and leeches that have only some fragments of the file completed. Both the types of peers will be responsible in sharing the completed fragments with the other peer community.

Compared to the generic client server approach the peer-to-peer network has multiple sources that can be utilized to acquire the required information. This availability of multiple sources can lead to complications in the selection of the most suitable peer to gather the information. As the source of maintaining track of the peers, the tracker is responsible for sharing the most suitable list of sources with the interested party[9]. Due to the lack of insight towards the requester, the BitTorrent algorithm will send a random list of peers that are active to the requester[10].

To ensure that the randomly selected peer list is providing the data in a desired manner, BitTorrent has implemented choking algorithms such as[7],

- Pareto efficiency
- BitTorrent choking
- Optimistic unchoking
- Anti-snubbing

At any given instance, the entire peer list returned by the tracker will not be used. A default size of 50 peers will be used in the data transfer mechanism. The unchoking algorithms will be running at the client side where they will be responsible towards discovering the most suitable peers from the peer list by trying to download the needed files from the tracker provided peer list. BitTorrent generally prefer using the BitTorrent choking algorithm and the optimistic unchoking heuristics to determine the quality of the peer.

In BitTorrent choking algorithm, the main criteria considered is current download rate from the peer[7] by considering the timespan of the past ten seconds. This can be seen as a biased approach to selected peers and puts an unfair burden towards the seeder. The other algorithm in use is optimistic unchoking which will randomly unchoke peers from the peer list to establish communications to determine the desirability of the peer.

Due to the scalability and the efficiency of the BitTorrent architecture, it is a suitable model to be used in the effort of caching and distributing the data directly from the clients instead from the server directly in an ad-hoc network where the clients can drop in and out of the network. We do not see many implementations that run over basic HTTP/HTTPS to run on the web browsers that have utilized this technology at an enterprise wide scale, but we can observe the implementation of the BitTorrent architecture at a large scale in standalone applications such as μ Torrent[19] to transfer large data files over the internet.

2.3 WebRTC for peer-to-peer communication

Previously to the introduction of the open source project, web real time communication (WebRTC) in 2012, the prominent way to generate a peer-to-peer connection between two devices were to open a direct dedicated port between the two devices. This approach was not suitable to be used over HTTP/HTTPS communication over the web browser as the ports 80 and 443 could not be used for other communication methods. Over the web content distribution via other method such as file transfer protocol (FTP) and architecture have been considered the approach to handling file sharing[11].

With the introduction of web real time communication, it has opened up new pathways to establish peer-to-peer connection between two web browsers to communicate over HTTP/HTTPS. At the time of writing, the WebRTC API is still being drafted by the World Wide Web Consortium (W3C), but the support for WebRTC has been provided for all major web browsers and operating systems such as, Google Chrome, Mozilla Firefox, Opera, iOS and Android. WebRTC provides support to connect two browsers without any 3rd party components to have real time data communications, such as audio/video communication and file transfer without the help of a server after the initial handshake has been completed[12] as displayed in Figure 5.

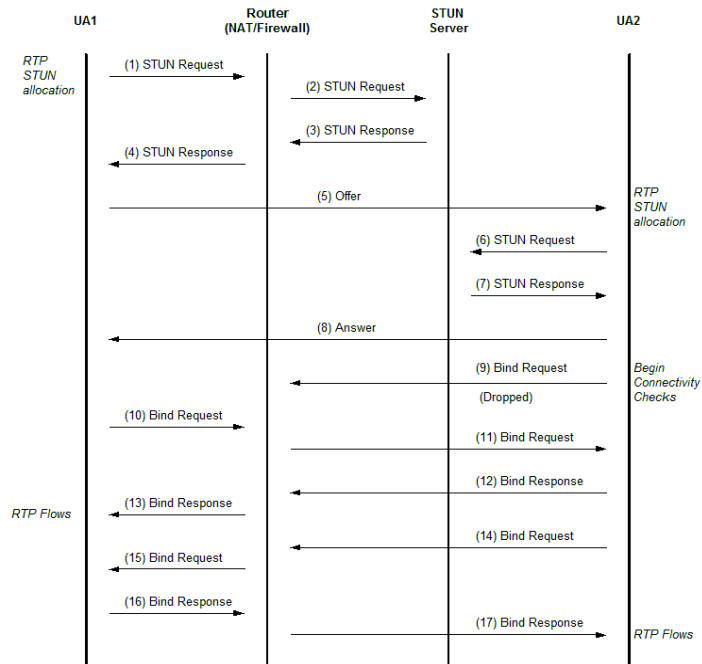


Figure 5 - WebRTC handshake between two clients

WebRTC utilizes IP to determine the location of the two end points of the connection. But due to the complex layout of the networks such as routing hierarchies and network address translations (NAT) for IP masking, WebRTC mainly relies on the following protocols to determine the hosts[12].

- Interactive Connectivity Establishment (ICE), for NAT-traversal for UDP-based media streams
- Session Description Protocol (SDP), to assist in setting up streaming media connections
- Session Traversal Utilities for NAT (STUN), a technique to discover whether or not the application is located behind a NAT, used by ICE.
- Traversal Using Relay NAT (TURN), extended version of STUN used to establish connections with processes with heavy firewalls.

This means that WebRTC is able to establish and communicate with remote parties even with NAT and firewall capabilities without compromising on the security aspects within the data transfer mechanisms. The basic technologies that are used for the connection establishment can be visualized in Figure 6.

Once the initial connections have been established, it is possible to transfer the data directly from the source to the destination in real time. The main use cases to the real time communication via WebRTC has been realized in terms of audio/video streaming and data transfer. In this research, higher focus will be directed towards the data transfer mechanisms to share the data between the peers.

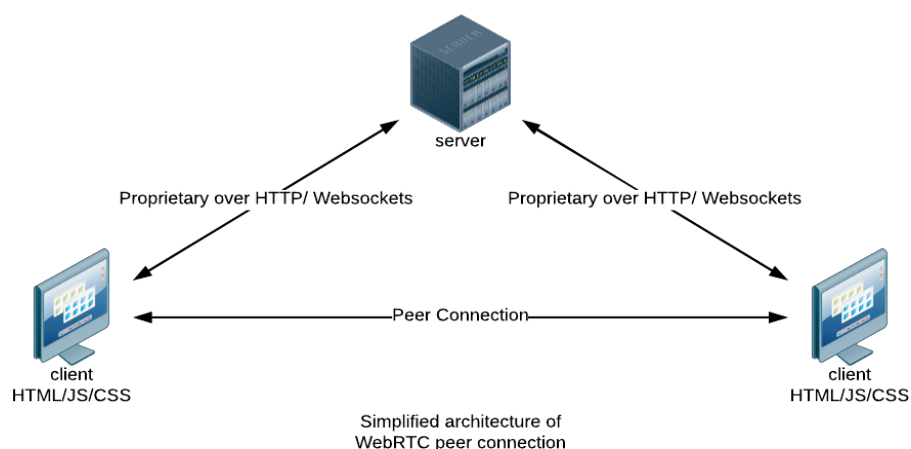


Figure 6 - Simplified architecture for WebRTC peer connection

2.4 Fusion of BitTorrent and WebRTC

It is possible to observe that we can improve the availability and increase the efficiency of sending data from the server to the client using the BitTorrent architecture characteristics. This approach allows the initially distributed data to be shared among the users without directly requesting all the information from the server.

This idea of torrent based sharing of data has led to the early implementation in 2003 by 'WebTorrent' which has been more directed towards the improvement of availability of the servers[13]. The WebTorrent implementation is done on top of Mozilla Firefox browser and Apache web server while having client and server side implementations to facilitate the data sharing process.

The main difference when it comes to sharing the data compared to the BitTorrent architecture implementations to the client server based model implementations is the difference in the size of the data that is being transferred[13]. Typically the size of the data fragment in BitTorrent is around 256KB which is significantly higher than that of web site content like HTML, CSS.

The WebTorrent implementation has handled this concern by bundling the content of the entire web page into a .tar archive before transmitting the data and reducing the minimum size of the transfer fragment to 16KB to improve the usability and the performance aspects of the BitTorrent infrastructure. But due to this implementation, it requires a considerable performance requirement at the client side with the need for a client side browser plugin. Also it has limited its usage to Linux based operating systems due to the compression format used[13].

Some of the limitations of 'WebTorrent' has been addressed with the advancements of web based peer-to-peer client connection mechanisms like WebRTC and has been come to realization with open source projects such as 'WebTorrent.io'. This means that the underlying protocol for data transferring is WebRTC rather than the standard TCP/uTP[14].

Other different implementations such as PeerCDN[14], which is currently owned by Yahoo, make use of WebRTC protocol for data transferring. But the main difference between PeerCDN and WebTorrent.io is that the elected peers for PeerCDN are “owned trusted centralized servers”, which was optimized towards a low latency downloads and fast peer discovery whereas the WebTorrent.io clients are ad-hoc users of the services without the need for a centralized authority.

It is possible to observe many alternate implementations of the WebTorrent.io architecture with different objectives in mind. Instant.io[15] can be considered one of the prominent offshoots of the implementation where the main focus is towards transferring files from one peer to the other via WebRTC protocol. Many other realization such as β Torrent, ZeroTube, PeerFast and FilePizza[15] can also be identified as significant implementation for this area.

Real time data transfer, ability to stream the data instead of needing to fragment into smaller components and the lack of compulsory requirement to compress the data prior to transfer has led to overcoming of certain problems of WebTorrent, but many other challenges such as optimized peer selection and improvements of piece selection are still concerns when considering about the data sharing aspects.

2.5 Conclusion

By the literature that was taken into consideration, it is possible to evaluate the advantages of the using peer-to-peer data exchange mechanism instead of client server architecture. Some of the critical aspects like bandwidth limitations can be taken as one of these directly observable advantages.

Peer-to-peer architecture talks only about connection of two peers, which cannot achieve the advantages that we need when considering distribution of the data. For this problem we can utilize the architecture that BitTorrent has proposed to decentralize the data and share it in an ad-hoc manner. This will improve the performance and the availability of the resources with the increase of the number of connected peers. But we still have to ensure the fairness of the data sharing between the peers which are not handled by BitTorrent.

Similar to the client server model the proposed architecture should also perform well in the web domain, without large changes to the existing model, which means that it should be working on standard HTTP/HTTPS protocols. To achieve both requirements, we have identified WebRTC protocol to connect the peers in the network without the need for any 3rd party applications.

Therefore, an architecture which will be consistent of a hybrid of client server and peer-to-peer model to share the data which utilizes WebRTC protocol with a modified BitTorrent algorithm to improve the selection of the peers and to share the data fairly within the connected peers can be the most optimal solution to overcome above mentioned issues with the least change to the existing infrastructure and with minimal cost.

3. Research Methodology

3.1 Introduction

In this chapter we will discuss on how the proposed architecture will be implemented, scaled and tested to simulate the behaviour and the expected benchmarks of a large scale implementation of such a system in the real world. Due to the limited resources available the scaling and performance benchmarking of the architecture will be executed in virtual environments which will be considered equivalent to the real behaviour.

3.2 Research Strategy

This research was conducted as an applied research, to determine the advantage of the application of this new architectural strategy while making necessary improvements to the BitTorrent algorithm to support this implementation.

The experimentation and the data collection will be carried out in a controlled environment where the nodes will be simulated to depict the natural behaviour of a client or a server. The networking interfaces will also be simulated and behave perfectly without any packet drops or firewall limitations. The network bandwidth limitations will be manipulated considering the peers and the servers accordingly using software based approaches.

3.3 Research Approach

The research will be consisting of two major components, namely the client and the server, similar to that of the generic client server architecture. The setup will consist of a ad-hoc network where the server will be the initial source of the data and the clients will be able to participate in the activities as needed. There will be a single server and one or more clients which will be active at any given moment in the test execution stage. The performance, latency and the data upload/download rates will be monitored and documented in both the server and the clients. As mentioned previously, the data that is documented will be mainly evaluated on their quantitative aspects rather than the qualitative attributes which will be covered by the existing infrastructure and protocol.

The test execution of this research will be simulated on virtual machines which will replicate the behaviour of the real clients and servers. Due to this approach we can gain few advantages over having physical machines and infrastructure available. Some of the main advantages that can be gained by this approach are,

- cutting down the cost of having number of physical devices to carry out the tests
- adding/removing nodes from the cluster can be done effortlessly
- monitoring activities can be directly configured in a single location
- network infrastructure configurations can be minimized
- problems related to firewall and data transmissions can be overlooked up to a certain degree

The concerns about the real life behaviour of the system can be mitigated when using this simulation approach as the proposed architecture is largely independent of the single client behaviour and as its having more weight towards the protocol and the behaviour of the participants will can be mimicked using the infrastructure simulation.

3.4 Research Design

This research high levely consists of three major activities. Connection establishment, system scaling and data gathering. The initial step would be to initialize the connections between client-server and client-client to initiate the data transferring. Once the setting up is complete, the system can then be scaled to accommodate a number of clients up to a maximum limit to determine their impact on the proposed architecture. Throughout the scaling process starting for the minimum number of client connections, the system will be monitored to determine the behaviour of both the clients and the server in this newly proposed architecture against that of the basic client server architecture which is having the same performance benchmarks.

3.4.1 Connection establishment

As previously mentioned the system will be running in a simulated environment throughout the implementation and the testing process. Therefore, during this task, we will focus mainly on incorporating the modifications to the BitTorrent algorithm and necessary development activities to establish the server.

The server and the clients will be both hosted in virtualized environments to have greater control over them in terms of resource and bandwidth limitations. The instantiated clients will be having limited performance attributes while being identical to each other to ensure the independence of the clients attributes to the outcomes of the test execution. The server will be built with Node.JS programming language and the clients will be Mozilla Firefox 61.0.1 instances as shown in Figure 7.

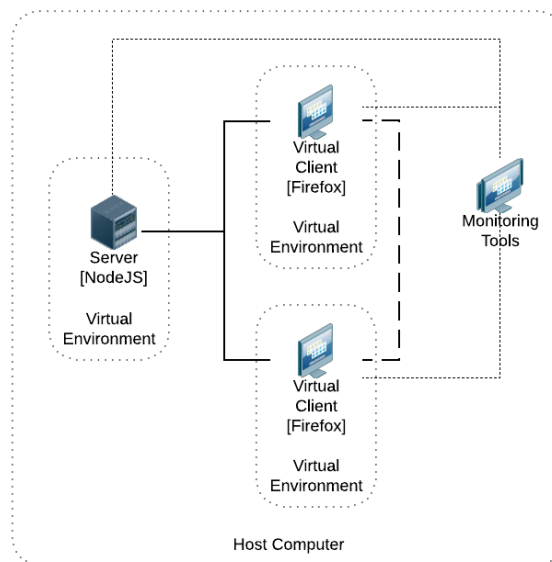


Figure 7 - Virtualized client implementation

Once the implementation is complete for the proposed architecture, its performance and behaviour will be measured using the monitoring tools which will be running on the host machine.

The connections between the server and the client would be mostly consisting of HTTP/HTTPS connections which will be running on the port 80 in the client and the servers. This communication will comprise of data related to the assets that the client requests, the metadata of the assets as .torrent files and list of other peers that contains the same asset and their publicly accessible IP addresses or client names.

At the initial state of the system, only the server will be the only location that the client can fetch the data from. Once a fragment of data is transferred to a client successfully, the server can determine that the said client can be a seeder for the data. This will allow other clients who are requesting the same information to fetch the data from the client who contains the information.

In the setup of a single server and two clients which will request the information from the server after a small delay should exhibit the following behaviour.

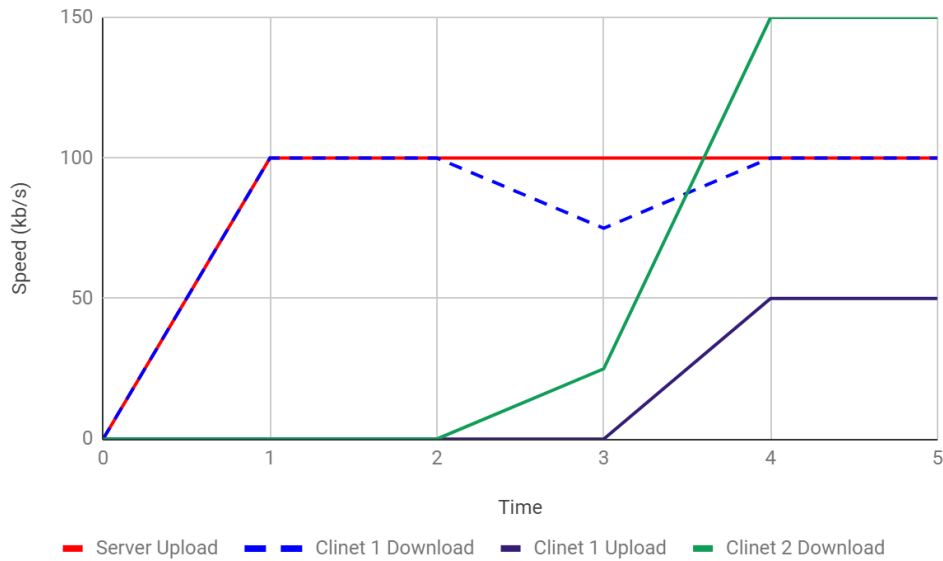


Figure 8 - Two client data upload/download rates

Assuming that the maximum upload rates of the server and client are 100kbps and 50kbps respectively and the request to fetch data for the second client start at interval 3, we can see that the data transfer rates experienced by the 2nd client can exceed that of the maximum upload rate of the server due to the data upload of the 1st client as seen in Figure 8.

Due to the nature of the BitTorrent algorithm, the asset will be transferred in fragmented sections such that once a fragment is completely downloaded at one client; it can start the seeding process as visualized in Figure 9. With the increase in the number of clients that are present, the download rate will increase up to a maximum which will demonstrate a logarithmic curve when a graph is constructed for download rate against the number of concurrent client connections.

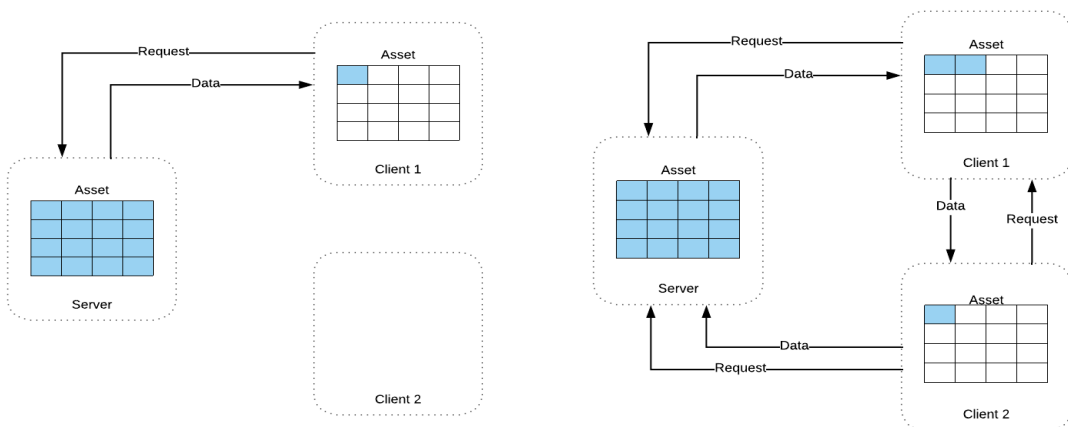


Figure 9 - Two client data upload/download rates

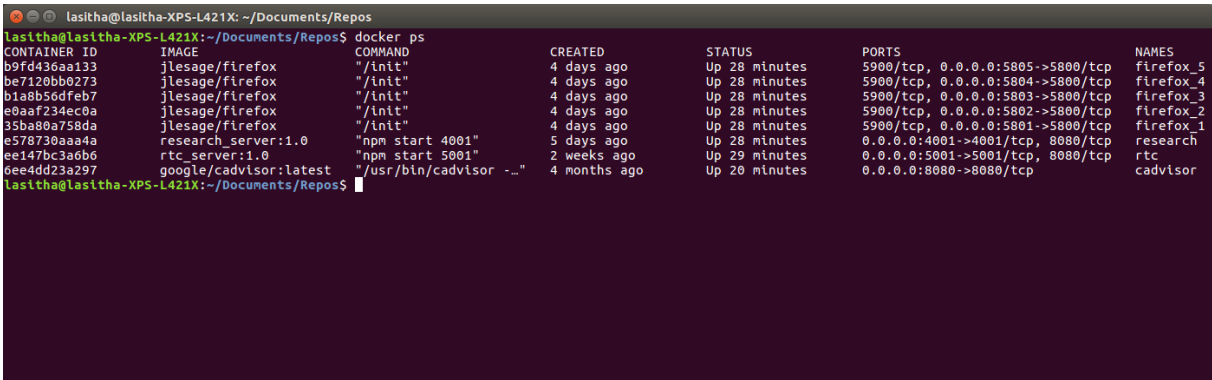
3.4.2 System scaling

During this part of the testing phase the number of clients will be increased keeping the server unchanged. By doing this, we can expect the number of request to the server to be increased.

When the number of clients is low we may not experience a deviation from the behaviour from the client server architecture and we may even experience a lower speed to that of the client server connection due to having additional steps to establish the WebRTC connection between the peers. But we when increasing the number of clients we can expect to see better performance and higher fault tolerance to ad-hoc behaviour of the clients.

Similar to the other instantiation of the clients, the scaling will be done with the help of virtualization. The scaling of the clients will be done with Docker technology which will allow full feature clients to be instantiated with the minimum performance and storage requirements while being able to be fully customizable and will behave similar to real computers. This will allow the monitoring process to accurately measure the usage of resources while minimizing the impact of scaling the system.

The clients will be running Mozilla Firefox 16.0.1 docker images during this experiment, and up to a maximum of 40 instances is expected to be running at the peak of this experiment. Some of the virtualized containers can be seen in below Figure 10, 11 and 12.



```
lasitha@lasitha-XPS-L421X: ~/Documents/Repos
lasitha@lasitha-XPS-L421X:~/Documents/Repos$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                               NAMES
b9fd436aa133      jlesage/firefox    "/init"            4 days ago         Up 28 minutes      5900/tcp, 0.0.0.0:5805->5800/tcp    firefox_5
b27120bb0273      jlesage/firefox    "/init"            4 days ago         Up 28 minutes      5900/tcp, 0.0.0.0:5804->5800/tcp    firefox_4
b1a8b56dfeb7      jlesage/firefox    "/init"            4 days ago         Up 28 minutes      5900/tcp, 0.0.0.0:5803->5800/tcp    firefox_3
e0aaf234ec0a      jlesage/firefox    "/init"            4 days ago         Up 28 minutes      5900/tcp, 0.0.0.0:5802->5800/tcp    firefox_2
35ba80a758da      jlesage/firefox    "/init"            4 days ago         Up 28 minutes      5900/tcp, 0.0.0.0:5801->5800/tcp    firefox_1
e578730aaa4a      research_server:1.0  "npm start 4001"   5 days ago         Up 28 minutes      0.0.0.0:4001->4001/tcp, 8080/tcp    research
ee147bc3a6b6      rtc_server:1.0      "npm start 5001"   2 weeks ago        Up 29 minutes      0.0.0.0:5001->5001/tcp, 8080/tcp    rtc
6ee4dd23a297      google/cadvisor:latest "/usr/bin/cadvisor -..." 4 months ago       Up 20 minutes      0.0.0.0:8080->8080/tcp             cadvisor
```

Figure 10 - Initialized docker virtual hosts

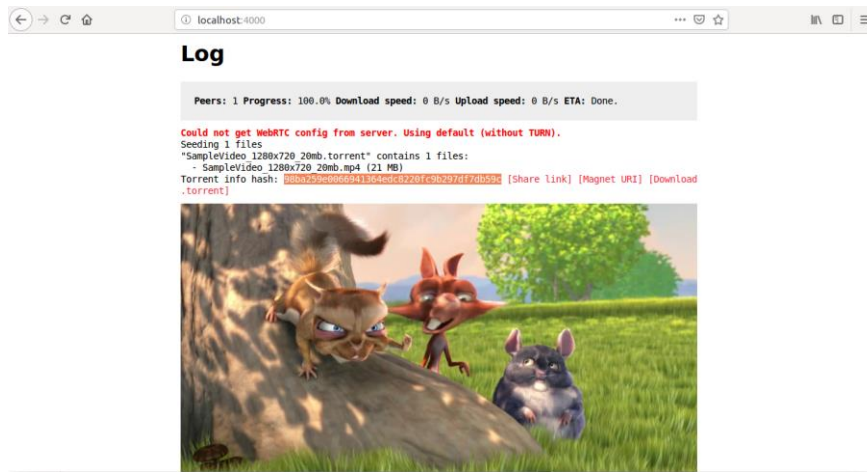


Figure 11 - Server and uploaded file

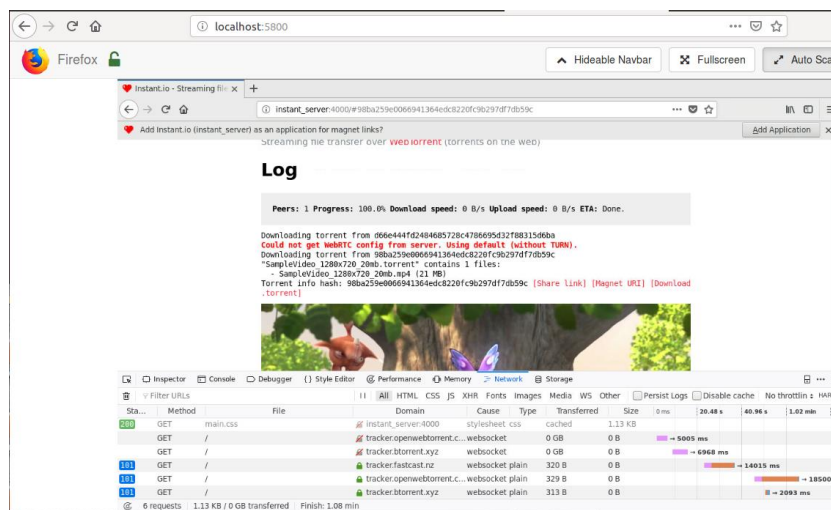


Figure 12 - Virtual Firefox client fetching the data from the server

3.4.3 Data Gathering

Data gathering will be conducted throughout the process mainly focusing on the visible aspects such as memory, CPU, latency and bandwidth usage in both the clients and the server but the main observable attribute will be considered the time taken for the data to be available at the client side. The main axis of which these attributes will be compared against is the number of actively participating clients in the system. This will be considered the control variable for the data gathering phase.

The data gathering for the interested attributes will be started at the steady state of the system, which will not be having any data in transit. The number of clients which request the data will be increased steadily up to the maximum while monitoring the behaviour of the system.

This statistics will be compared to that of the same as of the general client-server architecture under the same conditions to determine the performance difference of the two systems.

One of the main attributes of the newly developed system is the improvement of the download rate when the number of seeders is increased. This will also be taken into account in the data collection phase to determine the advantage that is gained when the number of peers is gradually increased.

3.5 Evaluation Plan

The criteria of evaluation of this defined research experiment are determining the improvement of the following criteria over the generic client-server model implementations of the same in identical environments.

1. Decrease in latency
2. Increase in speed of data arrival (download rate)

These criteria will be matched against the statistics collected in the different architectures with the number of the clients as the control variable, while all other variables will be kept constant such that they do not cause any effect to the data collection. The data will be gathered considering the client-server model, the basic BitTorrent based data sharing model against the improved peer selection model proposed to determine the impact.

To determine the success of the experiment, we expect to see an improvement in the speed at which the data will be available at the client side considering the improvements made toward the BitTorrent algorithm in terms of better peer selection and the fairness calculations. Due to the fairness attributes included into the algorithm, we can expect to see less biased peer selection and no bandwidth throttling in the peers.

Due to the nature of the web based environment, the data transfer can be expected to be of smaller size and with shorter time intervals compared to that of the BitTorrent algorithm. For this reason, the size of the data was selected to be that of an average size of a web page with such content, typically less than 10 megabytes in size.

In the BitTorrent algorithm implementation we can observe that a major component of data transmission is put upon the peers who are seeding data. Similarly we can expect to see a considerable participation from the peers in content seeding in the designed model. The biased peer selection of the BitTorrent algorithm is not suitable for this implementation as it would direct a considerable strain on the biasedly selected peer due to the overload of request from the other peers which request the data, taking up considerable upstream bandwidth from the seeder.

To avoid this scenario, the attributes of the uploaded and the downloaded data size has to be taken into account. This information needs to be utilized in the determining the logic in choking or unchoking of the incoming requests. This choking algorithm is optimized to maintain the optimal download to seed ratio[17] of 1:1 during the network communications.

To evaluate the fairness of the designed algorithm we compare it to that of the BitTorrent algorithm while considering the amount of data uploaded and downloaded as the measure. By setting up varying connection speeds we tried to simulate real life behavior of ad-hoc peer to peer network. Typically we looked towards maintaining a download to seed ratio close to that of 1:1 to ensure the fairness of the usage between the peers. This limitation was not forced upon the server as it can be considered the original source for the content.

4. Proposed Solution Details

The proposed solution which is running under the basics of the BitTorrent protocol behaves similar to that of a normal torrent. Therefore it will have a few milestones within its lifecycle. These steps include,

- Generating the torrent file and publishing
- File transmission
- File integrity verification

As the initial step in the process, the server will determine the files that need to be transmitted as a bulk. In this context, the web page that should be sent to the client and its content will be considered. For this generated torrent file, the information such as the files, the size of the content, the info hash and the list of peers that are available to seed the data will be created. Initially the server will be the only seeder available for the data transmission.

This torrent file then will be sent to the tracker where it will keep the torrent file saved. For the purpose of this research, the tracker was considered to be a 3rd party web server which was freely accessible for saving and transmitting the torrent files on demand.

This process of torrent file generation and saving can be performed prior to demand by the client to reduce the wait time. Once the torrent file has been processed by the tracker, it will be available for the clients to be accessed to initiate the respective file downloads.

Once the user requests for a file, the server will respond with the torrent file if available. The torrent file can either be transmitted as a downloadable file or a magnetic URI which will contain the details of the hosted torrent file. Due to the nature of the implementation, additional download of files which do not contribute to the web content is discouraged; therefore the magnetic URI approach was adopted.

Once the initial information has been provided by the torrent file, the client can begin the download process by generating the corresponding peer to peer connections provided in the torrent file via the use of WebRTC protocol implementation. These connections are named 'wires' in this context.

In the default BitTorrent algorithm, once the wires are identified, they are selected in a random order from the group. This approach does not identify the most suitable peer. This approach can be suitable for long running processes that BitTorrent is most suitable for, but

for dynamic environments such as web sites, the selection of the peer correctly should be done quickly as the client is waiting on the content to be loaded.

For this process, the return response time (RTT) was added to the wire. This will provide an indication of the distance from the client to the other seeders. Once the most suitable seeders are identified, the client can start the communication with them to retrieve the data. The RTT was selected as the optimal method that can be used to select the best peer as it provides the closeness of the seeder to the client. As the peer to peer connection is a direct link from the client to the seeder, this takes in to account of the physical location of the seeder as well as any network bottlenecks that maybe existing from the client to the seeder device. Thus RTT measurement will provide an accurate measurement of the speed at which the selected seeder can provide the data at. These RTT values will be continuously updated with each data request to always maintain the latest status of the seeders. We can maintain these values against the respective peers as in Figure 13 to produce a map to the best peer.

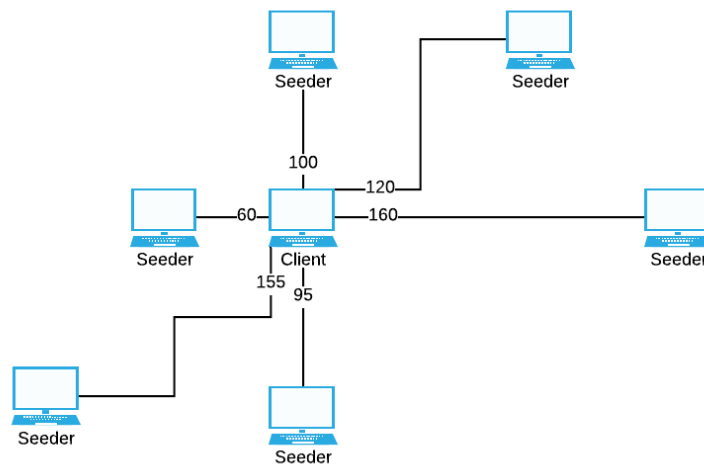


Figure 13 - RTT from client to seeder

Log

```

CurrentPeerId: 2d57573030332d534b7479377173576e647a77
Total Downloaded: 4.8 MB Total Uploaded: 524 KB
Peers: 4 Progress: 23.8% Download speed: 100 KB/s Upload speed: 0 B/s ETA: A few seconds remaining.
Uploaded: 0 bytes Downloaded: 502210 bytes Average speed: 180 KB/s Time taken: 2.794 s
PeerInfo:
Peer Id           RTT Downloaded AverageSpeed
2d57573030332d375355363669447661516647 103 98304 72 KB/s
2d57573030332d54633174615a754639676b43 46 114688 42 KB/s
2d57573030332d6e5549304d5067364f77794c 111 174530 84 KB/s
2d57573030332d722b6a516731674570747167 116 114688 45 KB/s
  
```

Figure 14 - Four clients RTT example

In the BitTorrent algorithm, once the download process has run for some time, the peer selection algorithm will consider the downloaded data size to determine the optimal peer. This process will make the peer selection algorithm biased toward that seeder, trying to forcibly download the content without any consideration for the seeder upload bandwidth. This means that the performance of the seeder will decrease. Even if the seeder's performance decreases, the biased peer selection algorithm will continue selecting the same seeder due to the previously uploaded data size.

This approach of biased peer selection will not be suitable for a web based solution as it throttle seeder's bandwidth making the other user's experience poor. This should be controlled in this web solution to ensure fair usage of the application for all the users.

The measurement of upload and the download volume was introduced to the seeder's side to allow the limitation of the upload by choking the clients who are requesting the information. All the clients will try to maintain the optimal upload to download ratio of 1:1 to maintain fairness throughout the use of the application. Once the client reaches the ratio of 1:1, they will be starting to choke the other peers. At the point where the client has downloaded more data, the ratio will reduce, unchoking the other peers allowing the upload to resume. The client will be unchoked in a periodical manner to allow the other peers to maintain the liveness of the client by a heartbeat connection, as depicted by Figure 15, while sending updates of the RTT status of the connected clients.

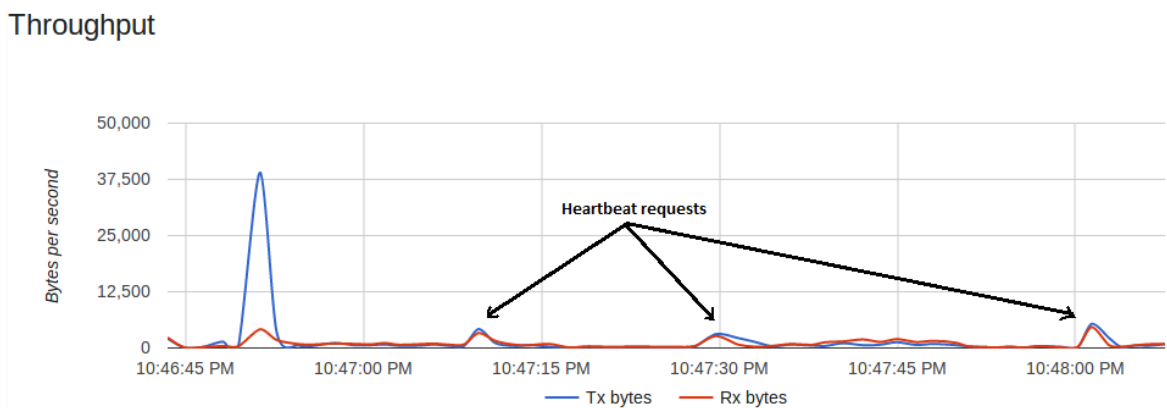


Figure 15 - Peer heartbeat

Once the download has been completed, the files will be verified against the info hash to ensure content has been correctly downloaded without any missing pieces or corruptions.

5. Evaluation and Results

The data collection was carried out considering the following implementation aspects for the system considering about the different architectures used.

1. Basic client server architecture
2. BitTorrent implementation for data distribution
3. Improvements for the BitTorrent algorithm

Considering the three implementations data transfer rates for each was recorded to determine the variation between the three.

5.1 Concurrent connection establishment

One of the main considerations can be taken as the impact of the system when multiple concurrent users are trying to access the same resource. The limitations of the server will have a significant impact on the data availability at the client side. To investigate the impact of the number of concurrent clients, the three implementations were hosted and data was gathered comparing the time taken for the download to be completed at the client side.

For this experiment, a file on side 2 megabytes was selected and the clients and servers were limited to a maximum bandwidth capacity of 100 kilobytes per second for the upload and the download rates. The number of concurrent clients was gradually increased to observe the behaviour of the download. The experiment was carried out multiple times to get an average reading for each instance.

During the data transmission we could typically see that the server is generally transmitting data at the maximum allowed rate throughout the entire transaction process in each of the architecture implementations as depicted in Figure 16.

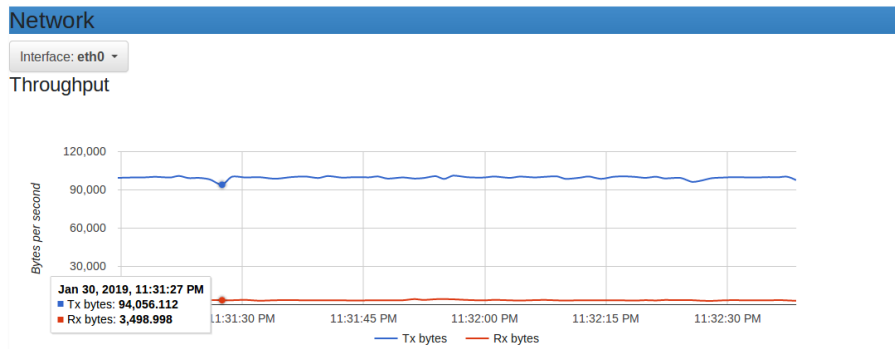


Figure 16 - Server data transfer activity

When considering the behaviour of the data transfer in the client side, we typically observe that the transfer is interrupted at regular time intervals in the client as shown in Figure 17, to allow other clients to download the data from the server in the client server model whereas the peer to peer architectures showed no such interruptions, indicating that the client was allowed to download the data at a consistent rate (Figure 18).

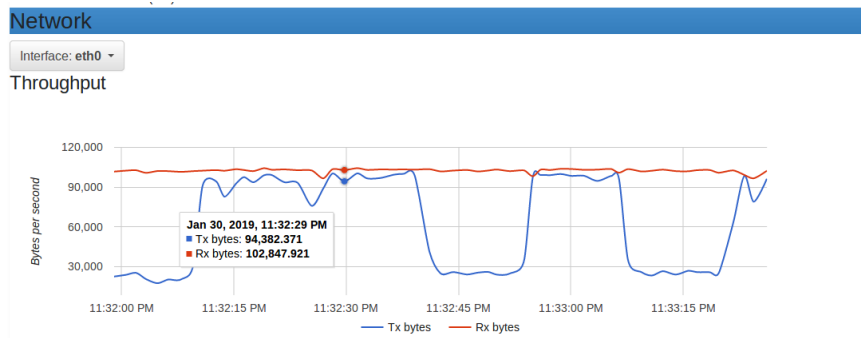


Figure 17 - Client data transfer activity of Client-Server model

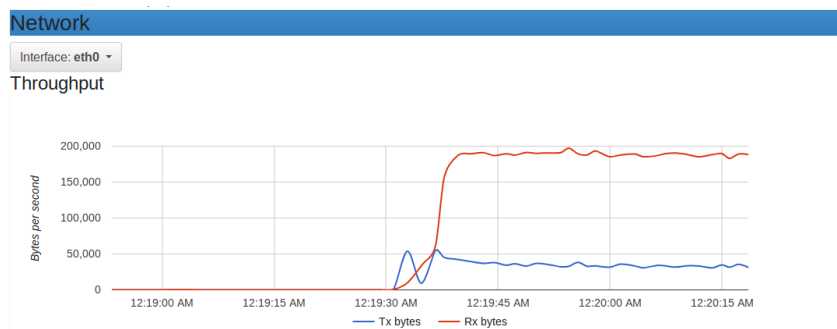


Figure 18 - Client data transfer activity of P2P model

Connections	Event (seconds)		
	Client Server	BitTorrent	Improvements
1	27.288	25	25.4
2	51.506	41.8	33.2
3	75.6	53.3	45.6
4	85.6	63.8	48.2
5	92	63.8	48.2
6	120	67.8	61.6
7	129.8	65.8	67.6
8	139.2	61.6	60.2
9	159	79.4	68.6
10	166	59.6	65

Table 1 - Time taken for parallel download

Considering the initial data collection statistics in Table 1 we can see that the client server architecture suffers highly when the bandwidth is limited. This can be directly related to the single source of data. As the server is forced to respond to each client individually, the response time is impacted in a linear manner.

When focusing towards the BitTorrent algorithm implementation, we can see that when multiple clients are introduced, due to the seeding nature of the clients, the time taken for the download is reduced to that of the basic client server implementations. After a few iterations we can observe the deviation for the time taken is reduced and bounces back and forth between a constant value of around 65 seconds. Similarly the improvement done to the peer selection algorithm shows a slight improvement compared to that of the BitTorrent algorithm and also stabilizes around 63 seconds when gradually increasing the concurrent peer count when observing the graphs of Figure 19 and 20.

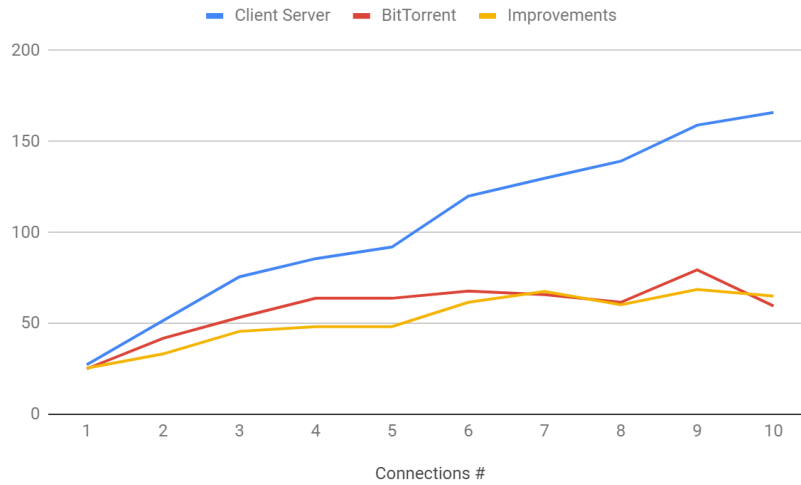


Figure 19 - Time taken for parallel download

When observing the graph generated with the time taken versus the number of concurrent peers, we can plot a linear curve for the client-server architecture implementation and a logarithmic curve for both the basic BitTorrent architecture and the improvements.

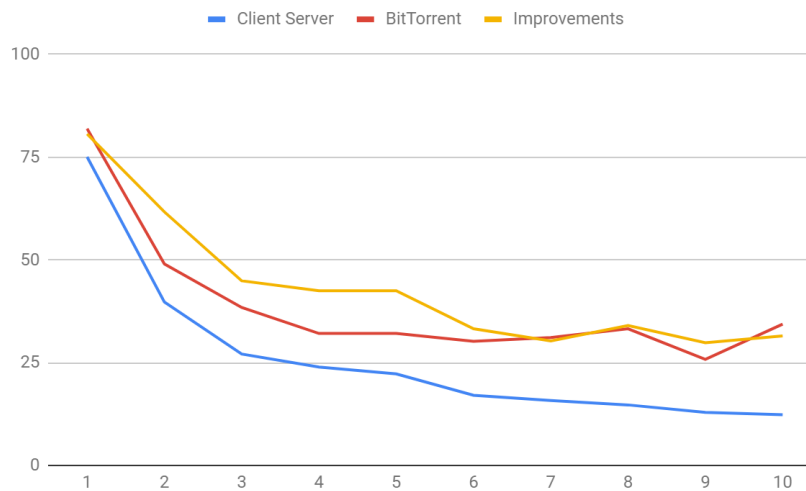


Figure 20 - Download rate vs number of concurrent peers

When taking into account of the rate of download for the clients, we can observe that the data rate rapidly drops for the client server architecture when new concurrent clients are introduced, whereas the BitTorrent and the improved algorithms stabilizes at a data transfer rate of around 31-33 kbps.

According to these observations we can determine that the performance of both the BitTorrent implementation and the improvements are superior to the basic client server architecture due to having multiple sources of receiving data. We can also observe a slight improvement in the altered BitTorrent algorithm over the BitTorrent algorithm. This can be traced towards the improvements in the peer selection algorithm where the peers who are having lowest latency is preferred over the random peer selection.

5.2 Data acquisition with multiple peers

In the basic client server architecture, once the information is transferred to the client the connection will be terminated between the two parties. Therefore it is unable to see any impact of introducing new peers once the initial download has been completed. We can expect to see no impact to the data transfer rates during this situation and the new peers to download the data with the maximum speed allowed by the server. In the BitTorrent algorithm implementation, the number of peers who have completed the download of the interesting data fragment will be able to seed that data to other parties. This means that the higher the number of available peers, the faster the data acquisition can be completed.

	Event (seconds)		
Connections	Client Server	BitTorrent	Improvements
1	27.2	25.44	25.18
2	27	15.24	13.64
3	26.5	9.4	9.54
4	26.9	8.72	8
5	26.1	7.54	7.8
6	25.6	6.24	6.5
7	25.8	5.72	5.86
8	27	5.74	6
9	26.8	5.14	5.68
10	26.2	4.8	5.32

Table 2 - Time taken with completed downloads

For this experiment, the maximum upload rates of the clients and the server has been limited to 100kbps while maintaining unlimited download for the clients to ensure to record the maximum allowed download rates.

As we can observe in the results that was gathered in Table 2, as expected the client server architecture is not affected by introducing new clients when a number of clients have completed the download process. But we can observe a big improvement when considering the BitTorrent and the improved algorithm implementations. This is due to seeding from the peers who have completed the download. The data that has been cached at the peers are uploaded when a new peer request the same data packet.

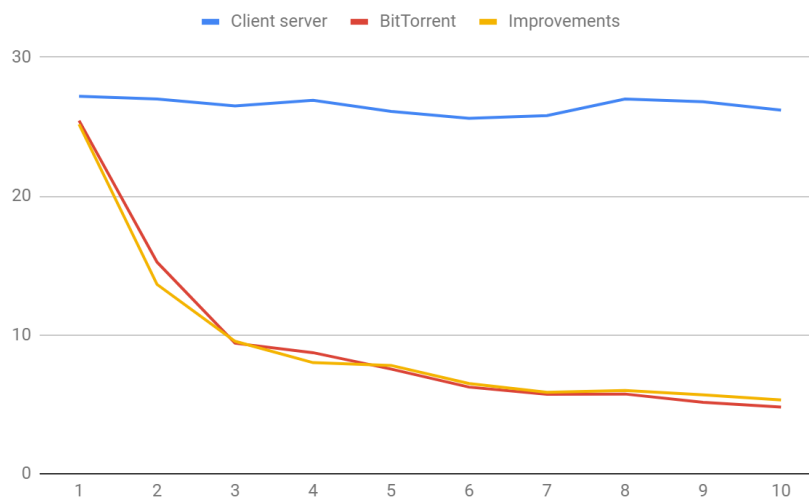


Figure 21 - Time taken with completed clients

When the experiment was carried out with a sample set of ten peers, we could see that the client server architecture was maintaining a linear non incremental graph for the addition of new clients as seen in Figures 21 and 22. As per the BitTorrent and the improved algorithm we can see a logarithmic curve.

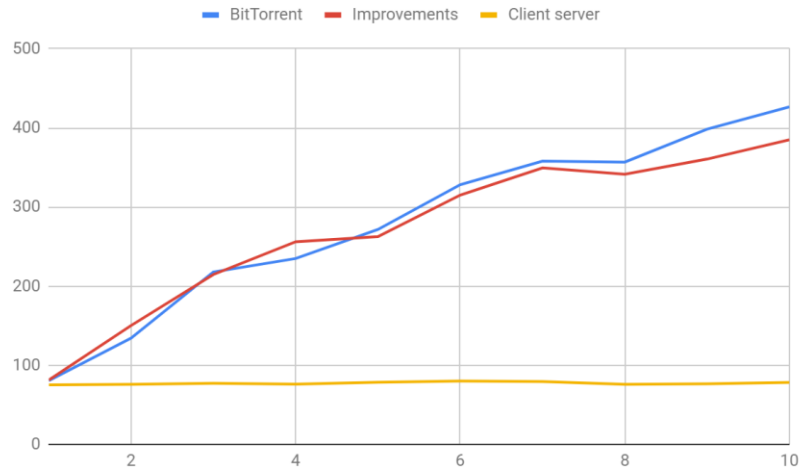


Figure 22 - Download rate vs completed clients

When observing the average download rates, we can determine that the client server architecture maintains an average rate of 77.6kbps rate over the entire experiment indicating no impact due to the increased peer count (Figure 23).

The basic BitTorrent algorithm and the improved algorithm both demonstrate improving average speeds with the introduction of new peers. We can observe a slight increase in speed at around 5-8% increase of average download speed when comparing the basic BitTorrent algorithm to the made improvements. This is associated to the fairness improvements performed at the new algorithm.

Log

```

CurrentPeerId: 2d5757303030332d655a7a694342715871614c74
Total Downloaded: 4.3 MB Total Uploaded: 2.3 MB
Peers: 5 Progress: 100.0% Download speed: 0 B/s Upload speed: 0 B/s ETA: Done.
Uploaded: 0 bytes Downloaded: 2107842 bytes Average speed: 291 KB/s Time taken: 7.264 s
PeerInfo:
Peer Id RTT Downloaded AverageSpeed
2d5757303030332d55666564784f427068414a57 780 616898 86 KB/s
2d5757303030332d4e70443239446d4e6c4c6554 1188 475136 70 KB/s
2d5757303030332d4a64626568736a796d524657 128 81920 72 KB/s
2d5757303030332d6a45446b6b66436b352f5651 1241 475136 72 KB/s
2d5757303030332d61466c7057634c6f6d326173 1220 458752 71 KB/s

```

Figure 23 - Downloading data from multiple peers

5.3 Determining fairness for peers

In the BitTorrent domain a peer is expected to maintain an upload to download ratio of 1:1 to be considered a good peer. This means that the peer is taking the optimal usage while contributing equally to the other peers within its lifecycle. In the generic BitTorrent algorithm this attribute is not critically maintained as it is not concerned with the fairness of the system. This means that the peer can be overwhelmed with other peer requests and its bandwidth can be used up. This behaviour needs to be removed when considering a web based system such as this to avoid client side frustrations.

To avoid this scenario the upload and the download amount has been introduced to the peer selection algorithm. The new algorithm now tries to maintain the optimal ratio of 1:1 for the upload and the download by each client by controlling the upload bandwidth. Once the ratio is unbalanced towards the upload, the peer will be choked to limit outbound data and to maintain its bandwidth. When the scale is balanced, the peer will be unchoked and will be allowed to seed the data as usual.

The experiment was carried out with the server, a seeder and a peer who is downloading the data. The server and the seeder were limited to a maximum upload capacity of 100kbps while the client download rates were set to unlimited rate.

As a fact we know that the server will be unchoked and will be providing the maximum possible download rates throughout the request, but the seeders will be choked depending on the ratio of upload and download.

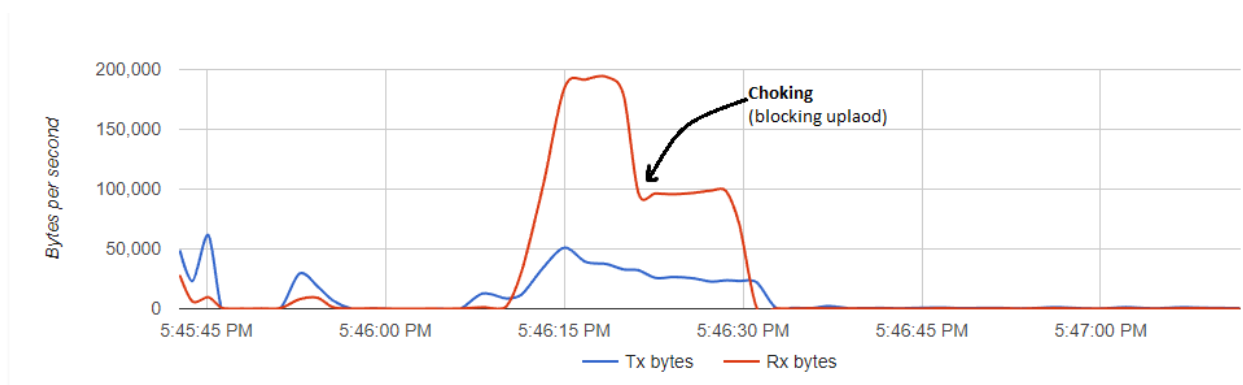


Figure 24 - Choking behaviour

In Figure 24 we can observe how initially the download process was backed by the server as well as the seeder allowing download speeds to reach far higher than allowed by the server only. Once the upload to download ratio has exceeded the 1:1 ratio, the seeder was choked thus reducing the download speeds to the maximum allowed by the server of around 100kbps.

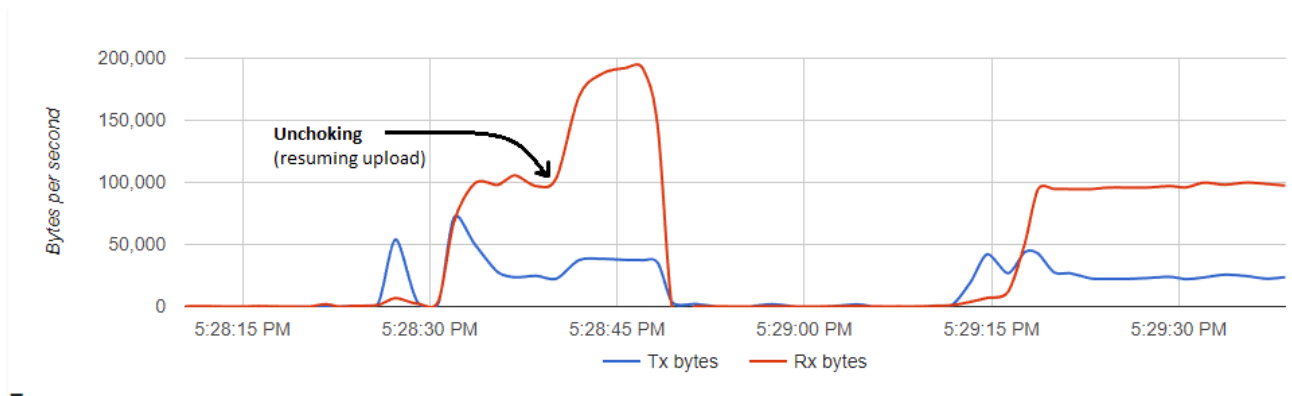


Figure 25 - Unchoking behaviour

Figure 25 shows the behaviour when the seeder gets unchoked allowing it to resume the upload which implies that the peer will be able to download the data at a much higher rate than by only using the server.

This choking/unchoking behaviour will make the architecture with the improvements to be slower than that of the BitTorrent architecture, when the seeders are choked, but it will ensure that the fairness is guaranteed for all the connected peers

6. Conclusion

Client-Server architecture is a well-established method for developing web based solutions throughout software development industry. Its versatility, scalability, ease of development and maintenance has become some of the key reasons that many tend to select the client-server architecture over other existing solutions. But this architecture scaling can only be done at the server side, by vertically or horizontal scaling approaches which will cost the hosting party a considerable amount of money. It also fails to solve some of the problems such as bottlenecks in the connections from the server to the client. Alternative solutions such as peer to peer connections can be utilized to solve some of these problems.

Throughout this research we have followed up on one of the extensions to the peer to peer architecture to overcome some of the limitations in the client server architecture. We have selected to utilize BitTorrent architecture based peer to peer system to provide this solution. Due to the very high scalability and the fault tolerance of BitTorrent it is an ideal alternative to host web solutions that are utilizing higher number of static content which are common for a number of people. In this proposed solution these content can be cached among the populus and can be redistributed via peer to peer connections over basic HTTP without the need for any additional software or hardware support. Due to the random peer selection behaviour and the biased selection of seeders in BitTorrent, the same algorithm cannot be used directly in a web based solution.

With the modifications that were done to the BitTorrent algorithm with the introduction of return response time (RTT) and the upload and download ratios, it was possible to create a suitable solution for the problem at hand. According to the analysis of the performance of the basic client server architecture against the modified BitTorrent algorithm implementation, it was possible to observe a considerable improvement in the download rates in the peer to peer based implementation. This can be traced to the download from the seeders where the problem of maximum speed of the channel from the server to client does not become the governing factor for the download speeds.

When comparing to the basic BitTorrent algorithm against the changes to the peer selection algorithm, we can observe that selecting the peers depending on their closeness to the client by the consideration of RTT has made minor improvements of around 4% to the download rate. But in the long run, we can see decrease in the speed of the download due to the fairness check implemented by the algorithm. This reduction was to be expected as unlike the biased

peer selection in the basic BitTorrent algorithm, web based implementation cannot completely use up the seeder's bandwidth.

In the process of this research, we have demonstrated how the peer to peer based architecture can be used in place of the client-server architecture to improve the data distribution performance of the server without the need for vertical or horizontal server scaling while adhering to the web based standard of HTTP/HTTPS without the need for additional software and hardware components. It has been demonstrated how the peer selection and the fairness calculations can be incorporated to the algorithm to provide a suitable method to discover seeders in a web based implementation.

6.1 Future work

This research was conducted on top of WebRTC communication protocol for the data transmission between the peers. This research can be extended to determine the impact for alternative peer to peer connection establishment protocols such as web sockets to determine their feasibility and potential improvements over using WebRTC.

Furthermore there can be security concerns related to having peer to peer based communication channels open between the peers and the server. One of the main concerns related to peer communication is the visibility of the peers to the other parties. Further investigations on how to ensure the peer anonymity can also be investigated in real life implementations of this architecture. Other approaches to reduce the data size such as on demand data compression before transmission can also improve the data speeds in this architecture.

References

- [1] D. Clark, "The design philosophy of the DARPA internet protocols", ACM SIGCOMM Computer Communication Review, vol. 18, no. 4, pp. 106-114, 1988.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.545.2055&rep=rep1&type=pdf>
- [2] R. Clarke, "Roger Clarke's 'Electronic Trading'", Rogerclarke.com, 2012. [Online]. Available: <http://www.rogerclarke.com/EC/ETIntro.html#L5>. [Accessed: 23- Oct- 2018].
- [3] R. Fielding, Architectural styles and the design of network-based software architectures. 2000.
- [4] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks", Communications of the ACM, vol. 49, no. 1, pp. 101-106, 2006.
- [5] R. Carter and M. Crovella, "Measuring bottleneck link speed in packet-switched networks", Performance Evaluation, vol. 27-28, pp. 297-318, 1996.
- [6] K. Lai and M. Baker, "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth", Usenix.org, 2018. [Online]. Available: https://www.usenix.org/publications/library/proceedings/usits01/full_papers/lai/lai.pdf. [Accessed: 24- Oct- 2018]
- [7] B. Cohen, "Incentives Build Robustness in BitTorrent", 2003.
- [8] D. Qiu and R. Srikant, "Incentives Build Robustness in BitTorrent", University of Illinois, 2018.
- [9] P. Maymounkov and D. Kademlia, "A peer-to-peer information system based on the xor metric", Cambridge, USA, 2003.
- [10] E. Cohen and A. Barabási, "Linked: The New Science of Networks", Foreign Affairs, vol. 81, no. 5, p. 204, 2002.
- [11] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies", ACM Computing Surveys, vol. 36, no. 4, pp. 335-371, 2004.
- [12] J. Abbink, K. Grigorjancs and J. Verdoorn, "Dynamic peer-to-peer game networks using WebRTC", 2013.
- [13] G. Sivek, S. Sivek, J. Wolfe and M. Zhivich, "WebTorrent: a BitTorrent Extension for High Availability Servers", 2003. [Online]. Available: <https://pdfs.semanticscholar.org/5122/1315351d5385dd4f7b6b1e31b40100e8d029.pdf>. [Accessed: 25- Oct- 2018].
- [14] "Webtorrent.io: WebTorrent - Streaming browser torrent client", [Online]. Available: <https://webtorrent.io/>, [Accessed: 24- May 2018].
- [15] "Instant.io - Streaming file transfer over WebTorrent", [Online]. Available: <https://instant.io/>, [Accessed: 24- May 2018].

- [16] "DiegoRBAquero/awesome-webtorrent-clones", GitHub, 2018. [Online]. Available: <https://github.com/DiegoRBAquero/awesome-webtorrent-clones>. [Accessed: 27- Oct- 2018].
- [17] μ Torrent Community Forums. (2019). "Seed / Peer count explanation". [online] Available at: <https://forum.utorrent.com/topic/14644-seed-peer-count-explanation> [Accessed 30 Jan. 2019].
- [18] google/cadvisor, *GitHub*, 2019. [Online]. Available: <https://github.com/google/cadvisor>. [Accessed: 20- Feb- 2019].
- [19] μ Torrent, 2019. [Online]. Available: <https://www.utorrent.com> [Accessed: 28- April- 2019].

Appendix

One client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		26.95	24	26	25.6	80	25.9	78.68725869
2		27.76	25	25.4	25.3	80.9486166	24.5	83.18367347
3		27.62	25	24.4	25.6	80	25.4	80.23622047
4		27.13	26	25.7	25.3	80.9486166	25.7	79.29961089
5		26.98	25	25.5	25.4	80.62992126	24.4	83.52459016
Average		27.288	25	25.4	25.44	80.50543089	25.18	80.93725179

Two client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		50	43.5	38	14.93	137.1734762	14.1	145.248227
2		51.14	42.5	28	13.2	155.1515152	14.3	143.2167832
3		51.58	42	36	16.9	121.183432	13.3	153.9849624
4		52.62	41	37	16.8	121.9047619	13.1	156.3358779
5		52.19	40	27	14.4	142.2222222	13.4	152.8358209
Average		51.506	41.8	33.2	15.246	135.5270815	13.64	150.1466276

Three client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		72	57	46.6	9.8	208.9795918	9.3	220.2150538
2		77	55	45	9.3	220.2150538	9.2	222.6086957
3		76	49	46.5	9.3	220.2150538	9.8	208.9795918
4		76	53	45.3	9.3	220.2150538	10	204.8
5		77	52.5	44.6	9.3	220.2150538	9.4	217.8723404
Average		75.6	53.3	45.6	9.4	217.9679614	9.54	214.6750524

Four client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		81	69	58	7.7	265.974026	7.5	273.0666667
2		87	66	47	9.1	225.0549451	8.4	243.8095238
3		90	63	42	8.7	235.4022989	7.9	259.2405063
4		82	61	40	9.1	225.0549451	8.4	243.8095238

5		88	60	54	9	227.5555556	7.8	262.5641026
Average		85.6	63.8	48.2	8.72	235.8083541	8	256

Five client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		89	69	58	8.7	235.4022989	6.8	301.1764706
2		97	66	47	7.3	280.5479452	7.8	262.5641026
3		90	63	42	6.8	301.1764706	8.5	240.9411765
4		89	61	40	7.7	265.974026	7.6	269.4736842
5		95	60	54	7.2	284.4444444	8.3	246.746988
Average		92	63.8	48.2	7.54	273.509037	7.8	262.5641026

Six client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		120	69	61	6.1	335.7377049	7.1	288.4507042
2		118	70	69	7.5	273.0666667	5.9	347.1186441
3		119	60	62	6.1	335.7377049	5.9	347.1186441
4		121	75	60	5.7	359.2982456	6.8	301.1764706
5		122	65	56	5.8	353.1034483	6.8	301.1764706
Average		120	67.8	61.6	6.24	331.3887541	6.5	315.0769231

Seven client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		134	70	67	6.5	315.0769231	5.9	347.1186441
2		128	68	69	6	341.3333333	4.7	435.7446809
3		127	70	72	5.5	372.3636364	6.7	305.6716418
4		129	59	75	5.5	372.3636364	6.2	330.3225806
5		131	62	55	5.1	401.5686275	5.8	353.1034483
Average		129.8	65.8	67.6	5.72	360.5412313	5.86	349.4880546

Eight client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1		140	60	52	6.4	320	6	341.3333333
2		141	55	57	5.7	359.2982456	5.9	347.1186441
3		138	68	65	6.3	325.0793651	6.3	325.0793651

4	145	64	62	5.1	401.5686275	5.8	353.1034483
5	132	61	65	5.2	393.8461538	6	341.3333333
Average	139.2	61.6	60.2	5.74	359.9584784	6	341.3333333

Nine client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1	145	89	62	5.3	386.4150943	6	341.3333333	
2	165	85	71	5	409.6	6.1	335.7377049	
3	166	74	68	5.1	401.5686275	5.8	353.1034483	
4	160	85	76	5	409.6	5.5	372.3636364	
5	159	64	66	5.3	386.4150943	5	409.6	
Average	159	79.4	68.6	5.14	398.7197632	5.68	360.5633803	

Ten client download rates

	Event	Load video	RTC	Improvements	RTC		Improvements	
Attempt								
1	176	52	65	4	512	5	409.6	
2	180	51	66	4.7	435.7446809	5.3	386.4150943	
3	172	65	61	5.5	372.3636364	5.1	401.5686275	
4	160	58	68	5.3	386.4150943	5.7	359.2982456	
5	150	61	63	4.5	455.1111111	5.5	372.3636364	
Average	166	59.6	65	4.8	432.3269045	5.32	384.962406	

Comparison between download rates and number of clients

	Event (s)		
Connections #	Client Server	BitTorrent	Improvements
1	27.288	25	25.4
2	51.506	41.8	33.2
3	75.6	53.3	45.6
4	85.6	63.8	48.2
5	92	63.8	48.2
6	120	67.8	61.6
7	129.8	65.8	67.6
8	139.2	61.6	60.2
9	159	79.4	68.6
10	166	59.6	65

Average speed			
	Client Server	BitTorrent	Improvements
1	75.0513046	81.92	80.62992126
2	39.76235778	48.99521531	61.68674699
3	27.08994709	38.42401501	44.9122807
4	23.92523364	32.10031348	42.48962656
5	22.26086957	32.10031348	42.48962656
6	17.06666667	30.20648968	33.24675325
7	15.77812018	31.12462006	30.29585799
8	14.71264368	33.24675325	34.01993355
9	12.88050314	25.79345088	29.85422741
10	12.3373494	34.36241611	31.50769231

	Client server	BitTorrent	Improvements
1	27.2	25.44	25.18
2	27	15.246	13.64
3	26.5	9.4	9.54
4	26.9	8.72	8
5	26.1	7.54	7.8
6	25.6	6.24	6.5
7	25.8	5.72	5.86
8	27	5.74	6
9	26.8	5.14	5.68
10	26.2	4.8	5.32

	Client server	BitTorrent	Improvements
1	75.29411765	80.50314465	81.33439237
2	75.85185185	134.3303161	150.1466276
3	77.28301887	217.8723404	214.6750524
4	76.133829	234.8623853	256
5	78.46743295	271.6180371	262.5641026
6	80	328.2051282	315.0769231
7	79.37984496	358.041958	349.4880546
8	75.85185185	356.7944251	341.3333333
9	76.41791045	398.4435798	360.5633803
10	78.16793893	426.6666667	384.962406