Comparative Analysis of File Transfer with Distributed Index using TCP versus QUIC
ECE/CSC 573:Internet Protocols

Aditya Kadole (akadole)
Aishwarya Karad (amkarad)
Ashwin Prasad (aprasad5)
Mounika Bachu (mbachu)
Niranjan Pandeshwar (nrpandes)

May 1,2020

Table 1: Component Breakdown

Component	**Component Weightage**	Aditya	Aishwarya	Ashwin	Mounika	Niranjan
High Level Design	0.1	40	30	10	10	10
Algorithm Development	0.25	15	20	25	25	15
Coding	0.35	25	15	10	25	25
Debugging	0.2	10	20	35	10	25
Report Writing	0.1	20	25	20	20	15
Per student aggregate Contribution	1	20.5	19.75	19.75	20	20

1 Introduction

1.1 Overview

Peer to Peer file sharing is an efficient method to share files compared to a client server architecture. We have implemented a P2P file sharing system using TCP and QUIC. The TCP implementation demonstrates the use of a distributed index of files and peers. Often file transfer is done using TCP which has several drawbacks such as higher latency due to the TCP handshake.
Quic has several benefits over TCP such as:

- **Reduced Connection Establishment latency:** Quic achieves 0-RTT for known servers.
- **Improved Congestion Control:** The current version has only marginal modifications over TCP and is based on TCP NewReno.
- **Multiplexing without head of line blocking:** It reduces head-of-line blocking issues but doesn't entirely solve them.
- **Connection Migration:** It allows for connection migration from wifi to LTE without connection renegotiation.

1.2 Definitions

QUIC: QUIC is a UDP based protocol and aims to provide a reliable end to end connection much like TCP but with much reduced latency. It does so by retransmitting lost data not at the level of UDP but rather at the level of QUIC. It can be implemented at the application level rather than at the OS kernel level making it easier to build on top of the existing protocol stack.

TCP: A connection-oriented transport layer protocol that ensures reliable delivery of packets, most of the traffic on the internet is TCP traffic.

Distributed Index:
An index of all the files and the peers they belong to from distributed sources is referred to as a distributed index. In this, the index is also distributed across the network. Using a distributed index, we can have several machines, all working on building overall index. This enhances both storage size and flexibility of index. In the report we have referred to this distributed index as "list of files" which is a csv file that every peer has. The format of this file is as follows:

"filename;IP addresses of Peer;Port number of Peer;Flag"
where the flag bit is set to 1 if that peer has that file and 0 if it doesn't.

Registration Server: This is an always on Server which has the list of active peers on the network. It has a well known IP address so that new users can send requests to it to join the P2P network.

Multi-threading: Multiple independent operations can be performed simultaneously. Multiple peers can request files to one server. The server sends files to both peers simultaneously.

2 Design

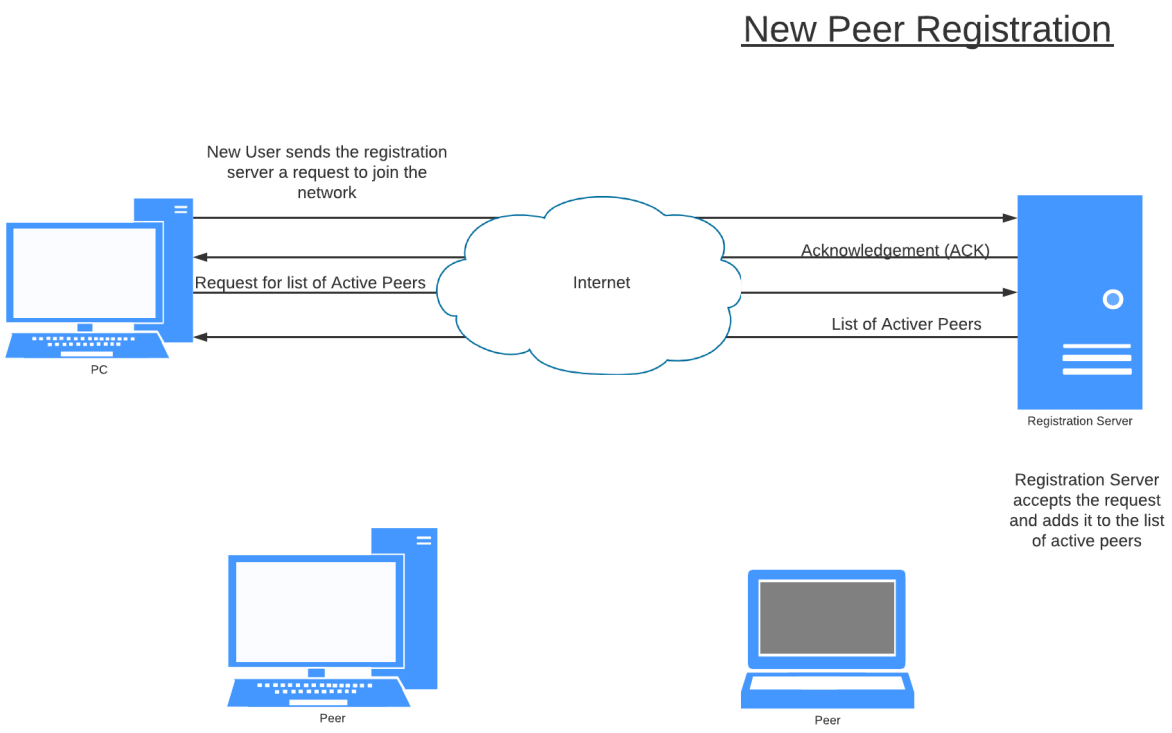

Figure 1: Registration of New Peers

Description: This diagram illustrates the series of messages sent by a new peer that wants to join the network.

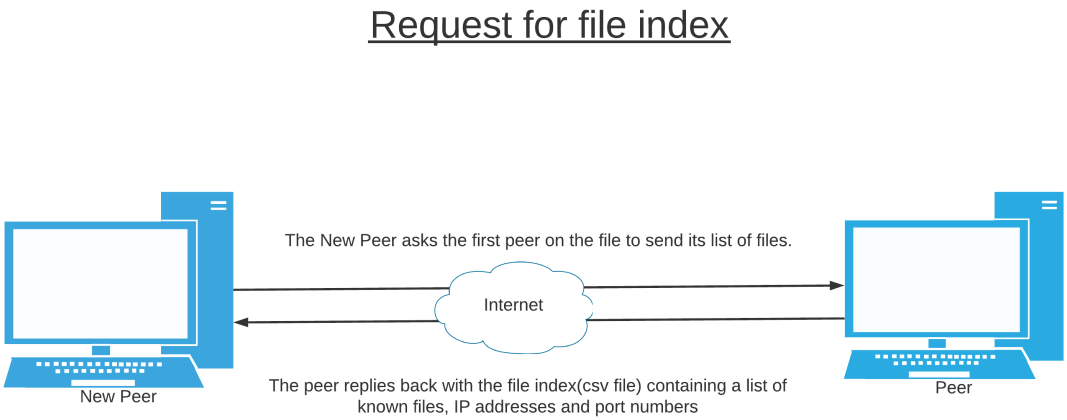

Figure 2: Requesting list of files

Description: This shows the interaction between a new peer and a peer it chooses to message from the list of active peers that it has (provided by the Registration Server). This interaction also occurs if a peer cannot find a file in the index of files it has and requests the RS for a fresh list of active peers.

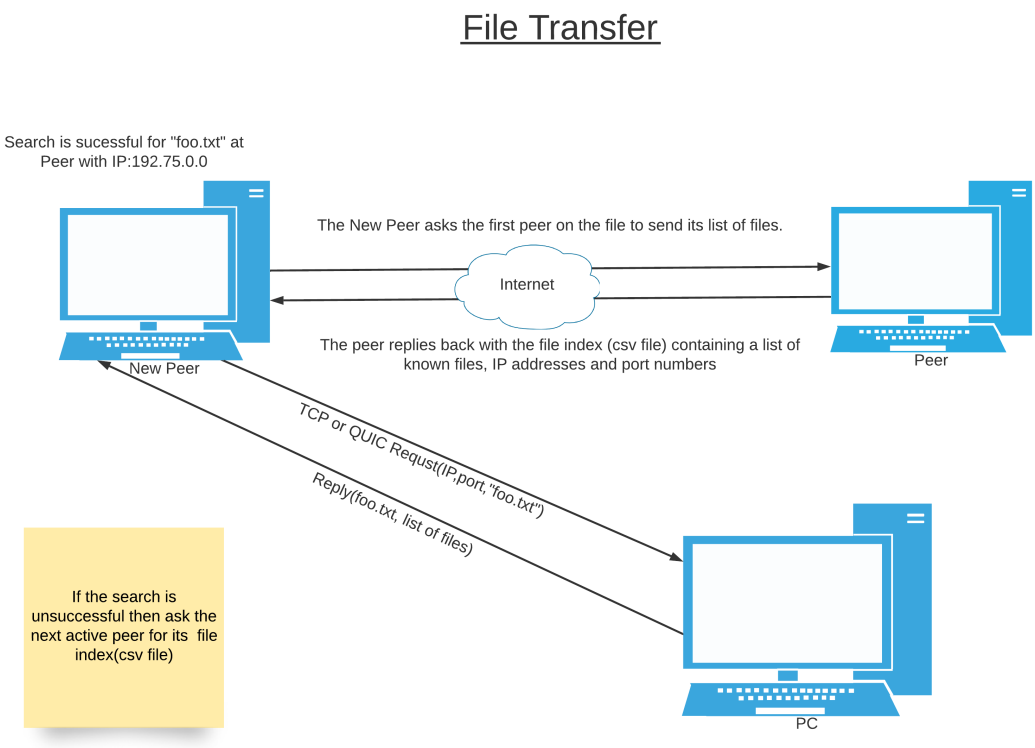

Figure 3: File Transfer

Description: This diagram demonstrates how files are sent between two peers. If it does not find a peer that has "foo.txt" in the list of files then it will traverse the list of active peers and send a request to another peer on the network for its list of files.

3 Implementation

We used **Python** to code our entire project.
Source Code Description for TCP P2P:

- **registrationserver.py:**
  1. This runs on the Registration server only.
  2. It listens for registration requests from peers who want to join the network.
  3. It can send a list of active peers to any peer that requests from it (requests are generated by client.py on other peers).
  4. It can also remove a peer from the network if it requests to be removed.
- **client.py**
  1. This runs on all the peers and is used when it wants to send requests for files or list of files from other peers using TCP sockets
  2. It can also send requests to the registration server for a list of active peers or to be removed from the network.
- **Server.py**
  1. This runs on all peers and it keeps listening for requests from other peers.
  2. If the peer requests for list of files then it sends the csvfile generated by csvfile.py.
  3. If a peer requests for a certain file that the peer has then it sends the file over a secure TCP connection.
- **csvfile.py:**
  1. The code runs every 30 seconds to discover the files on the peer.
  2. If it detects that a new file has been added in that peer's shareable file system then it adds it to the list of files and sets the flag bit to 1.
  3. If it detects that a file has been deleted by the peer then it sets the flag bit to zero in the list of files.

Source Code Description for QUIC P2P:

- **Quic_server.py:**
  1. The Quic server runs on every peer and keeps listening for client. requests generated by Quic_client.py.
  2. This QUIC server sends the files requested by peers in the network.
- **Quic_client.py:**
  1. The client code can connect with the registration server with requests to join/leave the network.
  2. It can request for a list of active peers from the registration server.
  3. The client can requests a list of files from each active peer one at a time.
  4. If the file that the client wants is present in the list, then it can request and receive the file from corresponding client.
- **RegisterationServer.py:**
  1. This code acts similar to the TCP Registration Server and provides the same features but handles QUIC packets.

4 Results and Discussion

Several experiments were conducted on the TCP P2P and Quic P2P network, the results have been tabulated below:

4.1 Throughput for TCP

File size(KB)	Time	Throughput(Kbits/s)
10	3.08739	24.27415
20	3.509	45.574125
40	3.6318	18.9787
80	5.333	122.54906
160	7.5466	173.21139
798	7.5943	860.8056
1596	7.6436	1710.14396
6383	7.71329	6778.77831
19148	13.7448	11412.31759
76592	34.3729	18253.90464

4.2 Throughput for QUIC

File size(KB)	Time	Throughput(Kbits/s)
10	0.48	170.6666667
20	0.95	172.4631579
40	1.64	199.804878
80	2.3	284.9391304
160	4.95	264.7919192
798	7	933.888
1596	7.3	1791.018082
6383	7.5	6971.938133
19148	11.8	13293.25559
76592	29.28	21429.01858

4.3 Results

As we can see from figure 4,5 Quic yeilds a higher throughput than TCP and takes lesser time.

5 Related Works and References

5.1 Related Works

- Quic chat application

5.2 References

- Peer to peer file transfer
- Decentralized file sharing
- Socket Programming
- aioquic repository
- aioquic documentation
- Python documentation
- How quick is QUIC
- Why Quic is not the next big thing
- Quic Protocol Performance in wireless networks

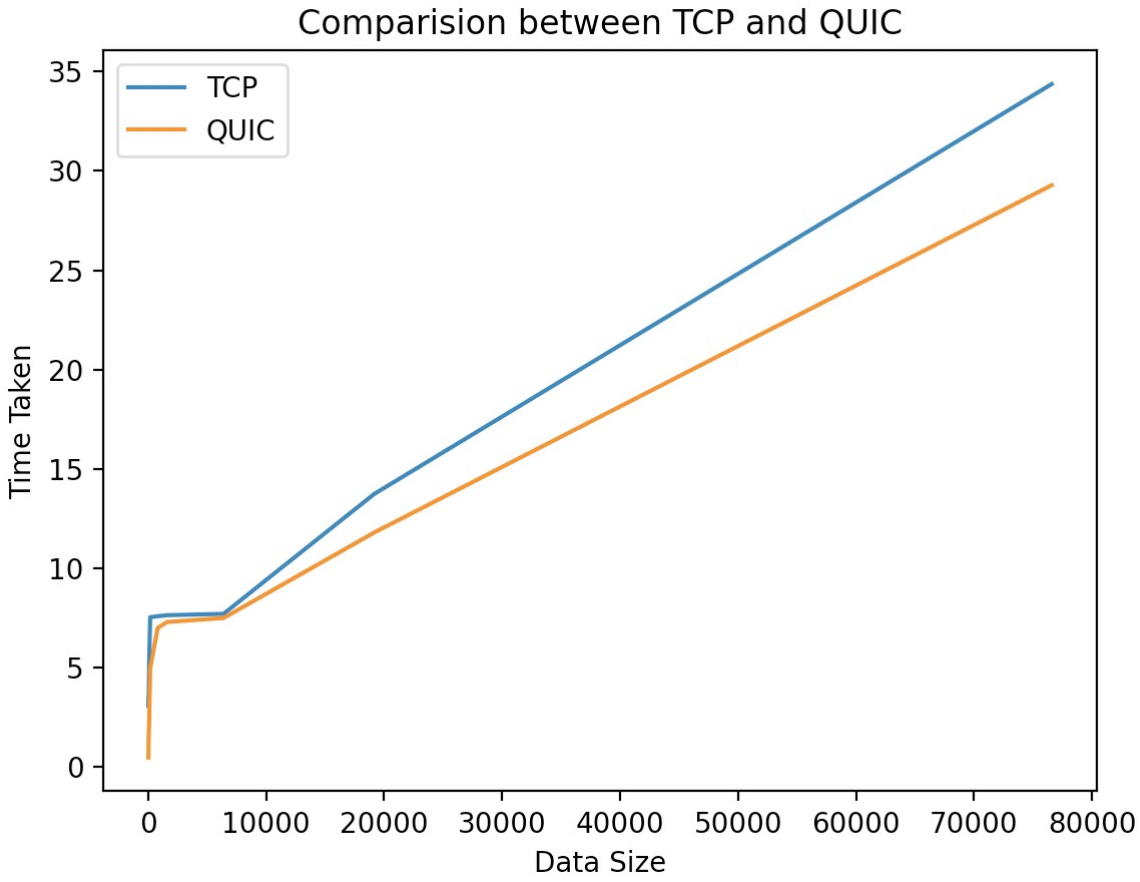

Figure 4: TCP vs Quic: Time Taken

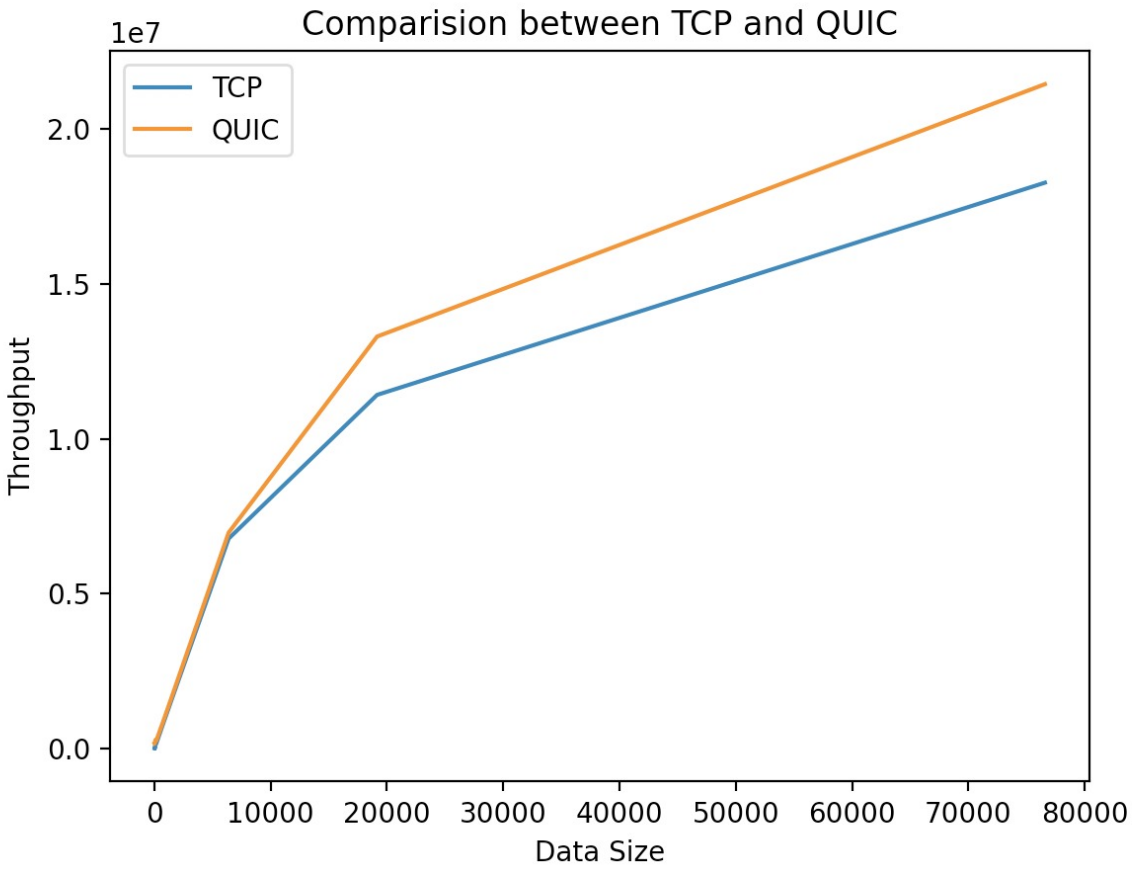

Figure 5: TCP vs Quic:Throughput