IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

ARCHITECTURAL ANALYSIS OF PEER-TO-PEER CLOUD STORAGE SYSTEMS

Pradeep Yadav, Rishabh Tiwari, Kaustubh Parab, Rashmi Pathak

Department of Master of Computer Application
Aditya Institute of Management and Research, Mumbai, India

Abstract: This study is about moving from traditional cloud storage to a new way of storing data called Peer-to-Peer networks. Traditional cloud storage is easy to use. It has some big problems. One company is in control of all the data. It costs a lot of money to keep all the servers running. We think we have found a way to make a system. We want to use something called Web Real-Time Communication and a special kind of encryption to let browsers store data safely on their own. Our main goal is to figure out how to move a lot of data on this new system and make sure the data is safe for a long time. We need to have a way to check that the data is still there which we call "Proofs of Retrievability" to make sure the data does not get lost over time. Peer-, to-Peer networks and cloud storage are the things we are looking at in this study.

Index Terms - P2P Storage, WebRTC, Decentralization, Data Security, Sharding.

I. INTRODUCTION

The way Peer-to-Peer (P2P) networking works has changed a lot. It used to be about programs like BitTorrent and eDonkey. Now it's more about using web browsers to share things. This change happened because of Web Real-Time Communication (WebRTC). WebRTC helps web browsers share audio, video and data without needing plugins.

This makes it easier for people to use applications.. It also creates a problem. The problem is that data only stays available if the person sharing it is still online. If they go offline the data disappears. Some projects, like KP8588 are good for transferring files. They're not good for storing data for a long time. This is because they don't have systems to keep data safe after the person sharing it goes offline.

To make a decentralized storage network we need to solve some big problems. We need to figure out how to send a lot of data over WebRTC. This means managing how data is sent and received. We also need to make sure that files are put back together correctly when they're received.

Another problem is that web browsers have limits on how much data can be sent once. This means we need to break files into pieces like 16 KB chunks. We then need to manage these pieces and fix any errors that happen. To make a system that lasts we need to get rid of choke points" and use Distributed Hash Tables (DHT) instead. We also need to add tools to check that people are keeping data safe over time. This can be done with cryptographic tools, like Proofs of Retrievability (PoR).

II. THE HYBRID ARCHITECTURE

The creation of a connection is tricky because it needs a central system to start with but then it wants to get rid of that central system. This mix uses a signaling server, made with WebSockets or Socket.io on a Node.js backend. The server helps two peers talk to each other at first.

They share some info like their network and device details. This info is in a format called SDP following the rules of JSEP.

Using this central server is a problem. If it fails or gets hacked the system can't make connections. This hurts the point of WebRTC, which is to have a direct connection between peers.

Another challenge is that most people use NAT and firewalls. These hide their IP addresses and show only one public address. WebRTC solves this with ICE. ICE tries all ways for two peers to connect and finds the best one.

To do this it needs two kinds of servers: STUN and TURN. STUN servers help peers find their IP and "punch a hole" through simple firewalls.

TURN servers are a plan, for stricter firewalls. They help relay traffic between peers. This costs a lot and can be slow.

So while these servers are necessary using them can make WebRTC slower and more expensive.

III. Engineering for Large-Scale Data

When you send files like a 1GB file over a browser-based Peer-to-Peer network it is really hard to do. The main problem is that different browsers have limits on how big a message can be. To make sure the file can be sent correctly on browsers, like Chromium and Firefox the application has to make sure each piece of the file is no bigger than 16,384 bytes.

This is because the WebRTC/SCTP protocol that is used under the hood cannot handle messages. So the application has to break the file into pieces and send them one by one. The developers have to use tools like the FileReader API to do this. They have to keep track of which pieces have been sent and which have not.

When the file is being received it is very important to manage the memory and buffer correctly. A special buffer, called receiveBuffer is used to put all the pieces of the file as they arrive. The system also keeps track of how much of the file has been received. It is very important that the receiver knows how the file is supposed to be before it starts receiving the file.

The receiver can only start putting the file together once all the pieces have arrived. Even though the DataChannel is reliable and sends the pieces in the order the application still has to have a way to track progress and handle interruptions. This is necessary to make sure the user has an experience.

The file is sent in pieces. The receiver has to put them back together. This is called assembling the file. The application has to make sure all the pieces are there before it can assemble the file. The Peer-to-Peer network and the browser-based application have to work to make sure the file is sent correctly. The developers have to write code to make sure this happens.

The receiveBuffer is used to hold all the pieces of the file until they can be assembled. The application has to keep track of how much of the file has been received and how much is still missing. This is a challenge for the developers. They have to write code that can handle all the things that can go wrong. The Peer-, to-Peer network and the browser-based applicati**ON HAVE TO WORK** to make sure the file is sent correctly and that the user has a good experience.

IV. Security and Trust

In a world where everything's all over the place we can't just trust computers to keep our information safe.

We have to be super careful with our info.

When our data moves from one place to another we need to keep it safe. That's why WebRTC tells us to use DTLS and SRTP. Think of DTLS and SRTP like locks that protect our data as it travels through the internet. Our data needs these locks to stay safe.

We also need to make sure only the right people can see our files. So we scramble our files on our devices before sending them. This way even if someone tries to snoop they won't be able to make sense of our files. Our files are like secrets. We keep them hidden.

We also need to ensure our files don't get messed up during sending. We use something called SHA-256 hashing to check if our files are okay. It's like a checklist that makes sure every single piece of our file is correct and nothing is missing or changed. Our files are. Double-checked.

Our data is important. We keep it safe, with DTLS, SRTP and SHA-256 hashing.

We are careful. Keep our information safe.

V. Performance Comparison

The Peer-to-Peer system is really good at dealing with problems because it does not have a Single Point of Failure. This means that if one node has a problem the whole system will still work. The Peer-to-Peer system is also very good at growing and changing because it can use all the nodes to do the work. This makes the system very strong. It can handle a lot of users. The Peer-to-Peer system is also very fast because the nodes can talk to each other directly which means that the data does not have to go through a lot of computers.

The Peer-to-Peer system is not perfect. It can be hard to know how well it will work because it depends on the nodes. If one node has a connection it can slow down the whole system. Most people use the internet at home and home internet connections are not always very fast. This can make it hard for the Peer-to-Peer system to work well. The system can handle a lot of users. The speed at which it can send data is not always very fast. It depends on the nodes and where they are located. The Peer-to-Peer system is very good at some things. It can be slow at times. The Peer-, to-Peer system has its advantages like being able to handle problems and grow easily. It also has some disadvantages, like being slow sometimes.

VI. Conclusion and Future Outlook

The change from a way to share files to a strong and private internet storage system is a big deal. This is because the old way was not good enough. At the center of this change is the problem of nodes leaving the network fast. This is called "peer churn". It can make it hard to get to the files. To fix this the system needs to use ways to make sure files are safe. One way is to use something called erasure coding. This breaks a file into pieces and makes extra pieces that can help put the file back together. This is better than making many copies of the file. With erasure coding you only need some of the pieces to get the file back. This makes the system very good at dealing with problems. It is like a kind of backup system called RAID 5. Even if many nodes go offline the file can still be recovered.

Making peer to peer storage good as big company systems requires checking all the time that the nodes are doing what they are supposed to do. This is done with codes called Proofs of Retrievability. These codes ask a node to prove that it has the file and that the file is okay without having to download the file. When you use these codes with a way to manage files called Distributed Hash Tables the system can find and check the files all around the world. By using these codes and math problems peer to peer cloud storage becomes a real solution that is as good as what big companies offer. P2P cloud storage is not an experiment anymore. It is an reliable way to store files. P2P cloud storage uses these codes and

math to make sure files are safe and can be recovered. This makes P2P cloud storage a good choice, for people who need a way to store their files.

VII. REFERENCES

- **Ali, A. (2001).** Macroeconomic variables as common pervasive risk factors and the empirical content of the Arbitrage Pricing Theory. Journal of Empirical Finance, 5(3), 221–240.
- **Basu, S. (1997).** The Investment Performance of Common Stocks in Relation to their Price to Earnings Ratio: A Test of the Efficient Markets Hypothesis. Journal of Finance, 33(3), 663-682.
- **Bhatti, U. & Hanif, M. (2010).** Validity of Capital Assets Pricing Model: Evidence from KSE-Pakistan. European Journal of Economics, Finance and Administrative Science, 3(20).
- **Dongdong, S. & Yan, W. (2020).** Blockchain Based Data Integrity Verification in P2P Cloud Storage. Western Sydney University Research Publications.
- **Gourley, J. & Anthoine, L. (2021).** Dynamic proofs of retrievability with low server storage. USENIX Security Symposium, 45.
- **Kumar, A. (2023)**. WebRTC Applications with Node.js and React.js. Medium Engineering Architecture.
- **Praveen, S. & Shafeeq, N.Y. (2019).** Effect of family environment on academic achievement of senior secondary school students. International Journal of Research in Engineering, IT and Social Sciences, 9(5), 322-326.
- **Rescorla, E. (2021).** Understanding WebRTC Security Architecture and NAT Traversal. Nabto Security Series.
- **Yuan, H. (2023).** BOSSA: A Decentralized System for Proofs of Data Retrievability and Replication. Haobo Yuan Research Repository.

.