**A**

**Project Report**

**On**

**“FILE HASHING MODEL USING TCP”**



**Department of Computer Science & Engineering**

# NATIONAL INSTITUE OF TECHNOLOGY PATNA

## University Campus, Bihar - 800005

**Submitted By:**

|  |  |
| --- | --- |
| **Name** | **Roll No.** |
| **Khushi Kumari** | **2006084** |
| **Sweta kumari** | **2006088** |
| **Saraswati Kumari** | **2006091** |

**Course Code: CSL5403 Course Title: Computer Networks Lab**

**Branch: B. Tech CSE**

|  |  |  |
| --- | --- | --- |
| **S.No. Contents** |  |  |
| **1. Introduction** |  |  |
| * + - * Hashing       * Hashing Components |  |  |
| * + - * Classification of Hashing Algorithms       * Working of Hashing Algorithms |  |  |
| * + - * Limitations |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **2.System Requirements** |  |  |
| **3. Flowchart/ Data Flow Diagram** |  |  |
| **4. Code** |  |  |
| * Code to implement hash value of a text file using client -server tcp model. * Code to implement hash value of an image file using client -server tcp model. |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **5. Output** |  |  |
| Output getting hashed value of text file |  |  |

## TABLE OF CONTENTS

Output getting hashed value of image file

### 6. Observation

MD5 vs.SHA-256

**7. Conclusion**

**8. Outcome**

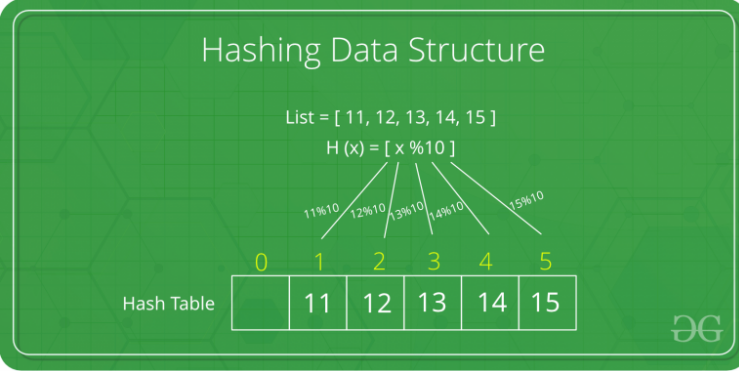
AIM OF THE EXPERIMENT

Implement a client server file hashing model using TCP. (Client send the file to server and return the hash value of the file)

## 1. INTRODUCTION

### Hashing

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.



Hashing is a popular technique for storing and retrieving data as fast as possible. The main reason behind using hashing is that it gives optimal results as it performs optimal searches

Suppose we want to design a system for storing employee records with phone numbers (as **keys**). And we want the following queries to be performed efficiently:

1. Insert a phone number and corresponding information.
2. Search a phone number and fetch the information.
3. Delete a phone number and related information.

We can think of using the following data structures to maintain information about different phone numbers.

Array of phone numbers and records.

1. Linked List of phone numbers and records.
2. Balanced binary search tree with phone numbers as keys.
3. Direct Access Table.

For arrays and linked lists, we need to search in a linear fashion, which can be costly in practice. If we use arrays and keep the data sorted, then a phone number can be searched in O(Logn) time using Binary Search but insert and delete operations become costly as we have to maintain sorted order.

Withbalanced binary search tree, we get moderate search, insert and delete times. All of these operations can be guaranteed to be in O(Logn) time.

Another solution that one can think of is to use a direct access table where we make a big array and use phone numbers as index in the array. An entry in array is NIL if phone number is not present, else the array entry stores pointer to records corresponding to phone number. Time complexity wise this solution is the best among all, we can do all operations in O(1) time. For example, to insert a phone number, we create a record with details of given phone number, use phone number as index and store the pointer to the created record in table.

This solution has many practical limitations. First problem with this solution is extra space required is huge. For example, if phone number is n digits, we need O(m \* 10n) space for table where m is size of a pointer to record. Another problem is an integer in a programming language may not store n digits.

Due to above limitations Direct Access Table cannot always be used. **Hashing** is the solution that can be used in almost all such situations and performs extremely well compared to above data structures like Array, Linked List, Balanced BST in practice. With hashing we get O(1) search time on average (under reasonable assumptions) and O(n) in worst case.  Now let us understand what hashing is.

**Why to use Hashing?**

If you observe carefully, in a balanced binary search tree, if we try to search, insert or delete any element then the time complexity for the same is O(logn). Now there might be a situation when our applications want to do the same operations in a faster way i.e. in a more optimized way and here hashing comes into play. In hashing, all the above operations can be performed in O(1) i.e. constant time. It is important to understand that the worst case time complexity for hashing remains O(n) but the average case time complexity is O(1).

**Basic Operations of Hashing:**

* **Hash Table:**This operation is used in order to create a new hash table.
* **Delete:**This operation is used in order to delete a particular key-value pair from the hash table.
* **Get:**This operation is used in order to search a key inside the hash table and return the value that is associated with that key.
* **Put:**This operation is used in order to insert a new key-value pair inside the hash table.
* **DeleteHashTable:**This operation is used in order to delete the hash table

**Hashing Components:**

**1)**[**Hash Table**](https://en.wikipedia.org/wiki/Hash_table)**:**An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.  In simple terms, we can say that hash table is a generalization of array. Hash table gives the functionality in which a collection of data is stored in such a way that it is easy to find those items later if required. This makes searching of an element very efficient.

**2)**[**Hash Function**](https://en.wikipedia.org/wiki/Hash_function)**:** A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in hash table. So, in simple terms we can say that a hash function is used to transform a given key into a specific slot index. Its main job is to map each and every possible key into a unique slot index. If every key is mapped into a unique slot index, then the hash function is known as a perfect hash function. It is very difficult to create a perfect hash function but our job as a programmer is to create such a hash function with the help of which the number of collisions are as few as possible. Collision is discussed ahead.

A **good hash function** should have following properties:

1. Efficiently computable.
2. Should uniformly distribute the keys (Each table position equally likely for each).
3. Should minimize collisions.
4. Should have a low load factor(number of items in table divided by size of the table).

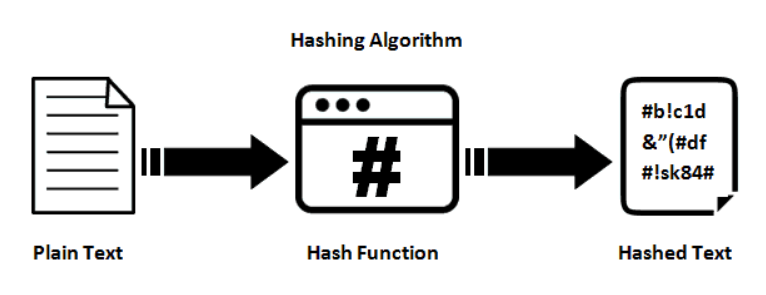
For example for phone numbers a bad hash function is to take first three digits. A better function is consider last three digits. Please note that this may not be the best hash function. There may be better ways.

**3) Collision Handling**: Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

* **Chaining:** The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
* **Open Addressing:**In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we examine the table slots one by one until the desired element is found or it is clear that the element is not in the table.

# Hashing Algorithms

Enter hashing algorithm, a fundamental part of cryptography, which refers to “chopping data into smaller, mixed up pieces which makes it difficult for the end user to go back to the original text/state”. A hash function is an algorithm that generates a fixed-length result or hash value from a specific input data. It is different from encryption which converts plain text to encrypted text and with the help of decryption, converts the encrypted text back to the original plain text. In the case of a hashing algorithm, plain text is converted into a hashed text through a cryptographic hash function, thereby making it difficult for hackers to make sense of it. (A hash length of 160 to 512 bits is good). But it doesn’t provide a way to go back to the original text.



So, if we have to ensure password security, hashing ensures that the passwords are hashed and stored in pairs along with usernames in the database table. When one logs in, the password typed is hashed and compared with the hashed entry from the database table. If there is a match, voila! The user is allowed to continue.

Hashtags can be used for password storage, integrity checks, digital signatures, message authentication codes. They can also come in handy for fingerprinting, file transfers, checksums etc.

## Hashing algorithm in action – How does it work?

We spoke about passwords and credentials at the start of the article. Now let us talk about file transfer. If one person (let us call him X) wants to transfer a file to another (let us call her Y). Without a hashing algorithm in place, the only way X can confirm the contents or the recipients would be to check in person with Y. But that would be cumbersome, and in a way, pointless in a busy, fast-moving yet highly insecure world. And if the message is long, the files are heavy or the mail contains multiple types, formats and numbers of attachments associated with it, this process will go for days on end.

But with a hashing algorithm, X can generate a checksum (a small-sized block of digital data derived from another block to detect errors during transmissions) for the specific file. Once Y receives the file and the checksum, she can use the same hashing algorithm on the received file. This would ensure that the correct file is sent by the correct sender, to the correct recipient.

## Types of Hashing Algorithms

### MD5 (MD stands for Message Digest)

One of the most commonly used yet amongst the most unsecure algorithms. When a password is converted into a specific pattern using this method, it is very easy to simply Google the hash value to get the original value. So, this is best avoided and, in fact, considered unsuitable for further use.

An example:

Input: An example of MD5  
Output (Checksum containing 32 digits hexadecimal number like the following): 6c30eeb06ce8eb66b7a65191272b9743

### SHA (Secure Hash Algorithm) family of algorithms

SHA-0, introduced in 1993, has been compromised myriad times. SHA-1, though a slightly improved version which has been used for Secure Socket Layer (SSL) security, has also been subjected to many attacks. SHA-2 is now recommended since it is more complicated. SHA-3 can be used by companies who are very serious about security.

An example:

Input: An example of SHA-1  
Output: 482ae821c8245e9545e3275cfec2e2657ccab6fb

### Whirlpool

It is a 512-bit hash function, derived from Advanced Encryption Standard (AES).

An example:

Input: An example of Whirlpool  
Output: 42fefc20dd412b5ad776271d1008ca65d1503a5acd384f3b4e3c8793ded11a0c3d853d721c6d23c37deeecc9b98765575c806099cec4a61b402b65b7a271bfd7

### RIPEMD family of algorithms:

It stands for RACE Integrity Primitives Evaluation Message Digest and was developed sometime in the mid-1990s. There are multiple versions like RIPEMD-160, RIPEMD-256 and RIPEMD-320. Since the output lengths keep increasing in the subsequent versions, the security coverage also increases.

An example:

Input: An example of RIPEMD-160  
Output: 033432770126267d6640cb35b1d7e1e75a78e7e5

### CRC32:

It is known as cyclic redundancy code and is commonly known for its spreading properties. It is also supposed to be a lot quicker leading to smooth file transfers and validations.

An example:

Input: An example of CRC32  
Output: 5c8e1a03

## Hashing Algorithm’s Security Limitations

Hashing algorithms are secure but are not immune to attackers. At times, a hacker has to provide an input to the hash function which can then be used for authentication. Multiple login attempts through brute force attacks can also be tried out till a match is found.

Since one exact input can have one exact output every single time, a typical, commonly-used password like ‘123456’ will be easier for a hacker to hash and gain unauthorised entry. Also, if multiple users are mapped to the same password, the hacker will be smiling all the way.

Another method called rainbow table attack where a hacker uses a large database of precomputed hash chains to crack passwords is common. Let us talk about the most commonly used password in the world – 123456. Let us consider Md5 hashing function. The way a rainbow table attack will work is as follows:

1. Pass the password (123456) through an MD5 hashing function to get: e10adc3949ba59abbe56e057f20f883e
2. Pass only the first few characters of the hashed value above (e10adc) to further get another re-hashed value: 96bf38d01b84aa16cf2bb9f55c61ac85
3. Repeat the above procedure until enough hashes are obtained in the form of a chain, starting from the initial plain text to the final hashed text
4. Store all of them in a table
5. Keep going through the list one at a time until a match is found

To counter such attempts, salting technique is used wherein further complexity is added to the hashed value to make it more difficult to crack the password. Here, random data is added to the input of a hash function to generate a much more complex output. Rainbow table mainly works on unsalted hash values, so this adds a further layer of security.

[Runtime Application Self-Protection](https://www.appsealing.com/rasp-security/) (RASP), which detects attacks on an application in real time, is a good practice to watch out for. With limited human intervention and a smart analysis of contextual behaviour of applications, better security is guaranteed. So, when any suspicious activity is detected, RASP would ensure to terminate a session or provide the relevant alerts to the users for further actions. And they do have an advantage over firewalls which just look at the perimeter of an application and don’t have much of an idea about what is going on inside an application.

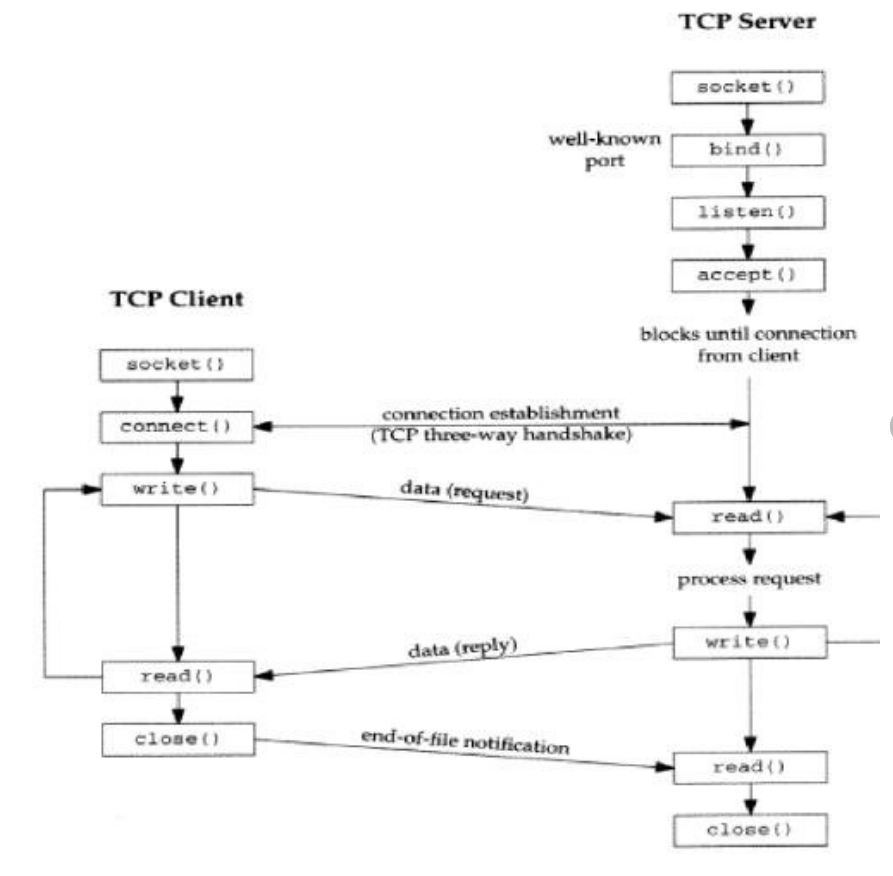
## 2. SYSTEM REQUIREMENTS

**Hardware Requirements:**

* 2 GHz x86 processor
* 256 MB of system memory (RAM)
* 100 MB of hard-drive space
* Monitor to display output
* Keyboard/Mouse for data input

**Software Requirements:**

* Eclipse IDE for Java Developers
* MS Word (Documentation)
* Text, Image ,video file



**Flow chart pf client server model**

**CODE:**

**Client.java program when input data is an image file**

package project;

//A Java program for a Client

import java.net.\*;

import java.io.\*;

public class Client

{

// initialize socket and input output streams

private Socket socket = null;

private FileReader file = null;

private BufferedReader input = null;

private DataOutputStream out = null;

// constructor to put ip address and port

public Client(String address, int port)

{

// establish a connection

try

{

socket = new Socket(address, port);

System.out.println("Connected");

// takes input from terminal

file = new FileReader("C:\\Users\\91808\\Desktop\\download.jpg");

input = new BufferedReader(file);

// sends output to the socket

out = new DataOutputStream(socket.getOutputStream());

}

catch(UnknownHostException u)

{

System.out.println(u);

}

catch(IOException i)

{

System.out.println(i);

}

// string to read message from input

String line = "";

try

{

line = input.readLine();

out.writeUTF(line);

}

catch(IOException i)

{

System.out.println(i);

}

// close the connection

try

{

input.close();

out.close();

socket.close();

}

catch(IOException i)

{

System.out.println(i);

}

}

public static void main(String args[])

{

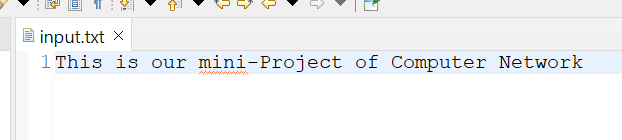
Client client = new Client("127.0.0.1", 5000);

}

}

**Download.jpg**





**Server.java**

package project;

//A Java program for a Server

import java.net.\*;

import java.io.\*;

import java.security.\*;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

public class Server

{

//initialize socket and input stream

private Socket socket = null;

private ServerSocket server = null;

private DataInputStream in = null;

// constructor with port

public Server(int port)

{

// starts server and waits for a connection

try

{

server = new ServerSocket(port);

System.out.println("Server started");

System.out.println("Waiting for a client ...");

socket = server.accept();

System.out.println("Client accepted");

// takes input from the client socket

in = new DataInputStream(

new BufferedInputStream(socket.getInputStream()));

String line = "";

MessageDigest digest;

// reads message from client until "Over" is sent

try {

digest = MessageDigest.getInstance("SHA3-256");

} catch (NoSuchAlgorithmException e) {

throw new IllegalArgumentException(e);

}

try

{

line = in.readUTF();

byte[] hash = digest.digest(line.getBytes("UTF-8"));

StringBuilder sb = new StringBuilder();

for (byte b : hash) {

sb.append(String.format("%02x", b));

}

System.out.println("hello : " + sb.toString());

}

catch(IOException i)

{

System.out.println(i);

}

System.out.println("Closing connection");

// close connection

socket.close();

in.close();

}

catch(IOException i)

{

System.out.println(i);

}

}

public static void main(String args[])

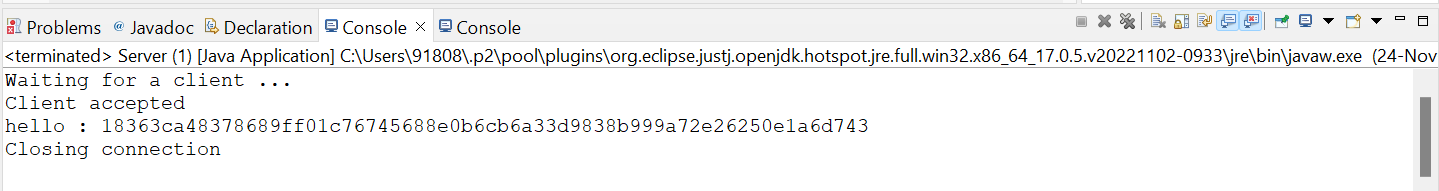
{

Server server = new Server(5000);

}

}

**Output:-**



**Comparing two hashing algorithms**

|  |  |
| --- | --- |
| MD5 | SHA-256 |
| **The output size is twice longer and the probability of collisions is lower.** | **SHA-256 is a bit slower than MD5, but it shouldn’t impact performances enough to not use it.** |
| **MD5 is the fastest cryptographic algorithm.** | **SHA-256 is about 20% slower.** |
| **MD5 returns 128-bits value.** | **SHA-256 returns a 256-bits value.** |
| **The collision probability is high.** | **The collision probability with SHA-256 is lower than with MD5.** |

## CONCLUSION

This mini project details the great potential that networking has. I was able to implement different broadcast and multicast routing protocols. The routing protocol is a routing algorithm that provides the best path from the source to the destination. In unicast routing, there is one source and one destination node i.e. point-to-point communication. The relationship between the source and the destination network is one to one. Each router in the path tries to forward the packet to one and only one of its interfaces. In broadcast routing, the network layer provides a service of delivering a packet sent from a source node to all other nodes in the network. In multicast routing, a single source node can send a copy of a packet to a subset of the other network nodes. Multicast routing is special case of broadcast routing with significance difference and challenges. In broadcast routing, packets are sent to all nodes even if they do not want it. But in Multicast routing, the data is sent to only nodes which wants to receive the packets.

This study helps me to realize the advantages and disadvantages of different routing protocols. Flooding generates duplicate packets and can cause Broadcast storm. The solution to this problem is Spanning Tree i.e. a tree which connects all the nodes and when the cost of traversal is minimum then it is called Minimum Spanning Tree. But there is a complexity associated with the spanning-tree based broadcast approach that is the creation and maintenance of the spanning tree. Multicast routing can be implemented by using Steiner Tree approach.

Thus I was able to execute different broadcast and multicast routing algorithms.

## LEARNING OUTCOME

Following are the learning outcomes of mine by this mini project that is “File hashing model using TCP”:

* After this experiment, I am familiar with different types of hashing algorithms.
* I am also able to write program for these algorithms.
* After this experiment, I am able to classify SHA-256 and MD5 algorithms.
* I am able to identify broadcast and multicast routing algorithms.
* After this experiment, I can apply my learnings when I want to develop a client server model using different algorithms.

## 9. REFERENCES

1. Tanenbaum S. Andrew and David J. Wetherall, Computer Networks (Vol. 5), Pearson Education Inc., 2011.
2. Forouzan A. Behrouz, Data Communications And Networking (Vol.4), McGrawHilI Forouzan networking series, 2007.
3. S.E. Deering and D.R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," ACM Transactions on Computer Systems, vol. 8, no. 2, May 1990, pp. 85-110.
4. S. L. Hakimi, "Steiner's problem in graphs and its implications", Networks, vol. 1, pp. 113-133, 1971.
5. B. K. Kadaba and J. M. Jaffe, "Routing to multiple destinations in computer networks", IEEETrans*.* Commun*.*, vol. COM-31, no. 3, pp. 343-351, Mar. 1983.
6. J. B. Kruksal, "On the Shortest Spanning Subtree of A Graph and the Traveling Salesman Problem", Proc*.* Amer*.* Math*.* Soc*.*, vol. 7, pp. 48-50, 1956.
7. Y. K. Dalal and R. M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", Commun*.* ACM, vol. 21, pp. 1040-1043, Dec. 1973.
8. T. Ballardie, CoreBased Trees (CBT version 2)MulticastRouting, Sept. 1997.
9. H. Takahashi and A. Matsuyama, "An Approximate Solution for the Steiner Problem in Graphs", MathematicaJaponica, vol. 24, no. 6, pp. 573-577, 1980.
10. P. Winter, "Steiner Problem in Networks: A Survey", *Networks*, vol. 17, pp. 129167, 1987.

.