Indian Institute of Technology, Bhubaneswar

SECURITY AND FORENSICS LAB 1

ASSIGNMENT-9

NAME - CHANDAN KESHARI
ROLL NUMBER - 24CS06022
BRANCH - CSE(MTECH)

Questions

CHORD Algorithm of File Sharing

1. Implement and simulate the CHORD algorithm of file sharing.
2. Simulate an eclipse attack on the third server which will be joining the network for the CHORD file sharing network.

Goal of Assignment:

The objectives of your assignment are:

1. **Implement and simulate the CHORD algorithm for file sharing:**
    - Develop a simulation of the CHORD protocol, which is a decentralized protocol designed to efficiently locate nodes storing specific data items in a peer-to-peer network.
    - This involves using consistent hashing to map both nodes and data onto an identifier space, ensuring data is distributed across nodes in a scalable way.
2. **Simulate an eclipse attack on the third server joining the network:**
    - Create a simulation of an eclipse attack, where an intruder attempts to control or isolate a specific node by creating false nodes surrounding it.
    - This manipulation can impact the data retrieval and storage functions of the targeted node.

These objectives will help you understand the implementation of the CHORD algorithm and the vulnerabilities of peer-to-peer networks to certain attacks, such as the eclipse attack.

Distributed Hash Table (DHT)

A **Distributed Hash Table (DHT)** is a data structure designed for a distributed environment, enabling efficient data storage and retrieval across a peer-to-peer network without relying on a central server. DHTs use a **hash table** structure to manage key-value pairs, where each key maps to a unique piece of data (or value). DHTs distribute data across many nodes in the network, and each node is responsible for a subset of the key space.
Key Features of DHT:

- **Decentralization:** DHTs eliminate the need for a central server. Each node has a specific responsibility in storing and retrieving data, making the system scalable and fault-tolerant.
- **Efficient Lookup:** A DHT can locate data in O(log N) time, where N is the number of nodes. This efficiency is achieved through nodes maintaining a small amount of information about other nodes to enable fast data lookups.
- Consistent Hashing: DHTs use consistent hashing, a technique that maps both nodes and data onto the same identifier space, typically a circular ring. This ensures balanced data distribution across the network.
- Fault Tolerance: If a node fails, data responsibility shifts to other nodes, so DHTs maintain resilience against node failures.

Basic Operations in a DHT:

- **Put(Key, Value):** Hashes the key to determine the node responsible for storing the value.
- **Get(Key):** Hashes the key, then routes the request to the node that stores the corresponding value.

CHORD Algorithm

The **CHORD algorithm** is a popular implementation of a DHT. It was developed to enable scalable, efficient, and decentralized data location in peer-to-peer networks. CHORD organizes nodes in a **ring structure** where each node is assigned a unique identifier (ID) generated through consistent hashing.

How CHORD Works:

- **Consistent Hashing and Ring Structure:**
  - Both nodes and data are assigned unique identifiers through hashing.
  - The identifiers are mapped onto a circular ring (e.g., a 160-bit identifier space), creating a structure where each data item can be assigned to a node in a predictable way.

- **Successor Nodes:**
  - Each node in CHORD has a successor node, the next node in the ring with a higher identifier. A key (data item) is assigned to its successor, i.e., the first node with an identifier equal to or greater than the key's hash.
  - The successor structure enables efficient lookup by ensuring that each node knows where its closest higher neighbor is.
- **Finger Table:**
  - To speed up searches, each node maintains a "finger table," a routing table with a small set of other nodes. Each entry in the finger table points to nodes at exponentially increasing distances around the ring.
  - For instance, a node might have entries for nodes 1 hop, 2 hops, 4 hops, etc., away, allowing it to skip large parts of the ring and reduce search time.

- **Efficient Lookup:**
  - To locate data, a node uses its finger table to quickly narrow down the node responsible for a given key, halving the search space with each hop.
  - The lookup operation has a time complexity of O(log N), where N is the number of nodes.

Joining and Leaving the CHORD Network:

- Joining: A new node finds its successor in the ring, adjusts its finger table, and updates relevant nodes' tables. The process ensures that the ring structure and data consistency are maintained.
- Leaving: If a node leaves the network, its data responsibility shifts to its successor, and other nodes update their finger tables accordingly.

Example Scenario with CHORD:

In a CHORD network of nodes with IDs ranging from 0 to 1023, a data item with a hashed ID of 250 will be stored on the node with the smallest ID greater than or equal to 250 (its successor). If nodes with IDs 200, 300, and 600 exist, the data will be stored on node 300.

Advantages of CHORD:

- ***Scalability:*** Efficient data lookup and distribution make CHORD suitable for large networks.
- ***Fault Tolerance:*** If nodes fail, data can still be found via alternate paths, as the finger table enables redundancy.
- ***Load Balancing:*** Consistent hashing helps balance data distribution, ensuring that no single node becomes a bottleneck.

CHORD Algorithm Overview:

- CHORD is a decentralized peer-to-peer (P2P) protocol that locates data across a network of nodes using consistent hashing.
- Each node and data item is assigned a unique identifier (usually a hash) to form a circular ID space.

Step-by-Step Guide:

- **Understand Consistent Hashing:**
  - Hash both nodes and data items to unique identifiers in the same identifier space.
  - Arrange these identifiers in a circular space (like a ring).
- **Set Up the CHORD Ring:**
  - Define nodes with identifiers (IDs) in a range (e.g., 0 to (2^m)-1, where m is the bit length of identifiers).
  - Place each node on the ring based on its hashed identifier.
- **Create the Finger Table:**
  - Each node maintains a finger table to efficiently find other nodes.
  - The table contains entries pointing to nodes at distances of from the node's ID (where i =0,1,2,...,m−1).
- **Implement Node Join:**
  - When a new node joins, it must populate its finger table and take over responsibility for some data items from the successor node.
  - Update finger tables of existing nodes to reflect the new node's addition.
- ***Implement Data Lookup:***
  - To locate data, hash the data's key to get an ID.
  - Use finger tables to hop through the ring until reaching the node that should hold the data.
- **Testing the CHORD Ring:**
  - Start with a few nodes, add data items, and attempt lookups to verify correct routing.
  - Check that data items are located in the expected nodes according to their hash IDs.

Question 1: Implementing and Simulating the CHORD Algorithm

.

Simulator Overview:

```
############################################################
************ CHORD Ring Simulator By Chandan ***************
############################################################

>>>>>>>>>>>>>>>> Menu <<<<<<<<<<<<<<<<<<<<
        1. Add Node
        2. Remove Node
        3. Store Data
        4. Look Your Data
        5. Display CHORD Ring
        6. Exit
```

Overview of Simulator app with multiple feature

Adding Node:

```
Enter choice from above options: 1
Enter id of node to add: 8
Node with id [8] added to the CHORD ring.
```

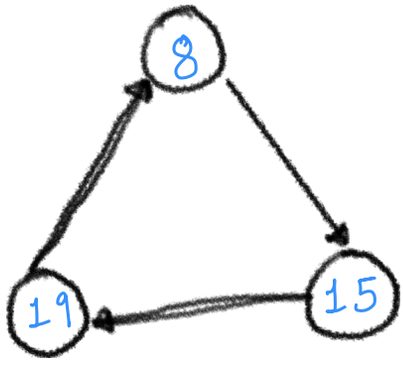
Adding a node with node ID → 8

Removing Node:

Initial structure of chord ring:

```
Node with id [8] :
        >>> Successor node: 15
        >>> Predecessor node: 19

Node with id [15] :
        >>> Successor node: 19
        >>> Predecessor node: 8

Node with id [19] :
        >>> Successor node: 8
        >>> Predecessor node: 15
```

Structure of chord Ring:



Removing node with id: 8

```
Enter choice from above options: 2
Enter id of node to remove: 8
Node with id [8] is removed from the CHORD ring and data transferred to its successor.
```

Removing node with ID → 8

Structure of chord ring now:

```
Node with id [15] :
        >>> Successor node: 19
        >>> Predecessor node: 19

Node with id [19] :
        >>> Successor node: 15
        >>> Predecessor node: 15
```

Structure of node now :

Storing Data:

```
Enter choice from above options: 3
Enter Node ID to store data: 5
Enter integer key: 7
Enter your data: hii
Stored key [7] at Node with id [15].
```

Storing data at node ID (5) with Key → [7]

Getting Data:

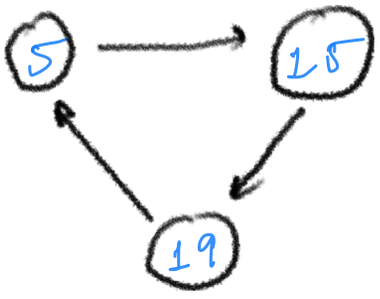
```
Enter choice from above options: 4
Enter Node ID to get your data: 5
Enter integer key: 7
Data for key [7]: {hii}.
```

Getting data stored at node ID (5) with key [7]

Structure of Chord Ring:

```
Enter choice from above options: 5

>>>>>>>>>>>>>> Every Node Structure in CHORD Ring <<<<<<<<<<<<<<

Node with id [15] :
        >>> Successor node: 19
        >>> Predecessor node: 5

Node with id [19] :
        >>> Successor node: 5
        >>> Predecessor node: 15

Node with id [5] :
        >>> Successor node: 15
        >>> Predecessor node: 19
```

Structure of chord Ring:



Exit:

```
Enter choice from above options: 6
Exiting...
```

Question 2: Simulating an Eclipse Attack on the CHORD Network

Eclipse Attack Overview:

An eclipse attack involves surrounding a target node with malicious nodes, isolating it from the legitimate network. This manipulation can alter routing and data availability.

Step-by-Step Guide:

- **Identify the Target Node:**
  - Choose the third server (or any specific node) in the CHORD network as the target for the eclipse attack.
- **Add Malicious Nodes:**
  - Create fake (malicious) nodes with identifiers close to the target node's ID.
  - This way, the target's finger table will point to these malicious nodes, isolating it from genuine nodes.
- **Modify Finger Tables:**
  - Adjust the target node's finger table to include the malicious nodes as its successors and neighbors.
  - This will redirect the target's queries and routing paths toward the malicious nodes, essentially "eclipsing" it.
- **Observe the Effects:**
  - Try to look up data from the target node and observe that queries are routed incorrectly or fail to retrieve the correct data.
  - Note how the target node becomes disconnected from the legitimate network due to the attack.
- **Test with Different Scenarios:**
  - Add additional nodes to the network and vary the position of malicious nodes to see how robust the CHORD network is against this type of attack.

Simulator Overview:

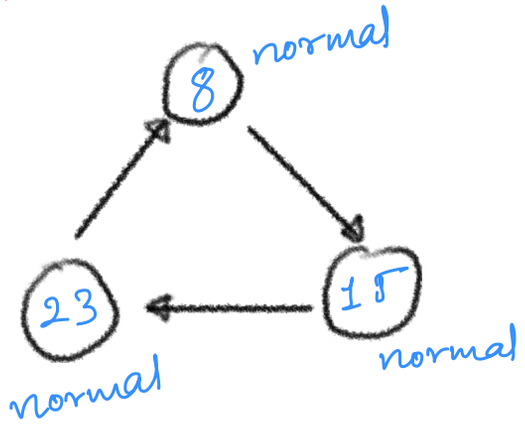
```
############################################################
************** Simulator with Eclipse Attack ***************
############################################################

>>>>>>>>>>>>>>>> Menu <<<<<<<<<<<<<<<<<<<<
        1. Add Node
        2. Remove Node
        3. Store Data
        4. Look Up Data
        5. Display CHORD Ring
        6. Simulate Eclipse Attack
        7. Exit
```

Overview of Simulator with Eclipse attack

Structure of Chord ring before attack:

```
>>>>>>>>>>>>>> Every Node Structure in CHORD Ring <<<<<<<<<<<<<<

Node with id [8] :
        >>> Successor node: 15
        >>> Predecessor node: 23
        >>> Status: Normal

Node with id [15] :
        >>> Successor node: 23
        >>> Predecessor node: 8
        >>> Status: Normal

Node with id [23] :
        >>> Successor node: 8
        >>> Predecessor node: 15
        >>> Status: Normal
```

Structure



Storing Data at Node ID: 23

```
Enter choice from above options: 3
Enter Node ID to store data: 23
Enter integer key: 27
Enter your data: hlo
Stored key [27] at Node with id [8].
```

Storing data at node with id 23 & key 27

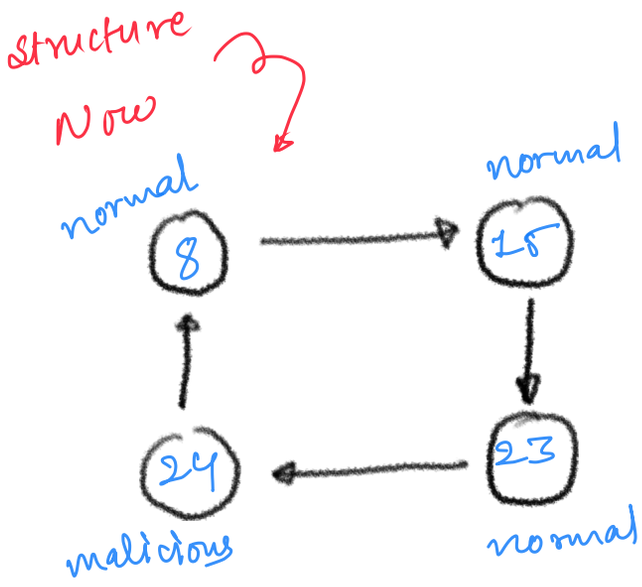
Simulating Eclipse Attack:

```
Enter choice from above options: 6
Enter Target Node ID for eclipse attack: 23
Enter number of malicious nodes to add: 1
Simulating eclipse attack on Node 23 with 1 malicious nodes.
Node with id [24] added to the CHORD ring. [Malicious]
```

Here, one malicious node 24 is added to the chord ring to targed node with ID → 23

Structure of Chord ring after attack:

```
>>>>>>>>>>>>>> Every Node Structure in CHORD Ring <<<<<<<<<<<<<<

Node with id [8] :
        >>> Successor node: 15
        >>> Predecessor node: 24
        >>> Status: Normal

Node with id [15] :
        >>> Successor node: 23
        >>> Predecessor node: 8
        >>> Status: Normal

Node with id [23] :
        >>> Successor node: 24
        >>> Predecessor node: 15
        >>> Status: Normal

Node with id [24] :
        >>> Successor node: 24
        >>> Predecessor node: 23
        >>> Status: Malicious
```



Not able to retrieve data from normal node:

```
Enter choice from above options: 4
Enter Node ID to look up data: 23
Enter integer key: 27
Data for key [27]: {Fail to retrieve the correct data !!}.
```

for node 23, we are not able to retrieve data as it got attacked by malicious node 24