Design and Implementation of Peer-to-Peer E-Mail System

K. Ramesh, Venkateswarlu K., R. Chandra Sekharam
Department of Computer Science and Engineering,
National Institute of Technology – Warangal – 506004,
Andhra Pradesh, INDIA
srikram2002@yahoo.com

Abstract

E-mail has become the most widely used communication facility for all institutions and individuals in the present era. Existing E-mail systems employ a server-centric design in which the user is critically dependent on the mail server. Insecurity due to mail sever failures is a major problem faced by many users. Design and implementation of a secure Peer-to-Peer E-mail facility is addressed in this work. Locating the specific computer that stores the emails of a specific user is the critical design requirement for which a solution is offered through the implementation of Chord protocol, which maps an E-mail entity onto a specific node. The architectural design and implementation also provides for better confidentiality. The prototype implementation has been observed to provide a good scope for improved anonymity. We present our prototype implementation and discuss the scope of future extensions.

1. INTRODUCTION

The present day E-mail systems employ a sever-centric design for handling and storing e-mails. If the mail server is down the user can neither receive nor access mails in such a case. This traditional server-centric approach has many other drawbacks like, less reliability due to sever failure, high demand on Server storage space, high performance demand on servers, excessive message duplication due to mailing lists, etc.

In this paper, running Internet e-mail application as a server-less application over a Peer-to-Peer (P2P) Chord ring is explored for two reasons. First, a P2P design [10] can eliminate the single point of failure. It can more easily store large messages by providing orders of magnitude of more storage, and eliminate server stress by distributing server functionality to millions of peers, and can handle mailing lists more efficiently. The second reason is simply because it is a challenging P2P design problem. Successful design implementation of this P2P E-mail system can be easily extended to many other Internet applications.

Considering the prototype design of P2P E-mail system one can choose a distributed hash table (DHT) substrate, such as Chord [1], CAN [7], Tapestry [9], or Pastry [8]. In our work Chord protocol has been chosen. Many distributed peer-to-peer applications need to determine the node that stores a data item [3]. The Chord protocol solves this challenging problem in a decentralized manner. It offers a powerful primitive: given a key, it determines the node responsible for storing the key's value.

Overview of the architecture, design details of Data and Procedures, prototype implementation details, conclusions and future scope are presented in the subsequent sections.

2. ARCHITECTURE

The proposed P2P E-mail system architecture is shown in the Figure 1. It consists of system nodes and Email User Agents (UA). The system nodes are the computers in a chord ring whereas the UAs use these system nodes to access the e-mail system. The UAs are the mail reader programs that are run by the users. The UA must have the IP address of at least one of the system nodes in the chord ring to access this e-mail system.

The role of the system nodes is to provide persistence for messages that are in transit from sender to recipient. This E-mail system uses the lookup service of a Chord for providing persistence of messages. Chord provides the following lookup service: The application supplies an arbitrary key (an element in the chord ring) and a variable k. The lookup service returns to the application the k active nodes in the chord that are the closest to the key.

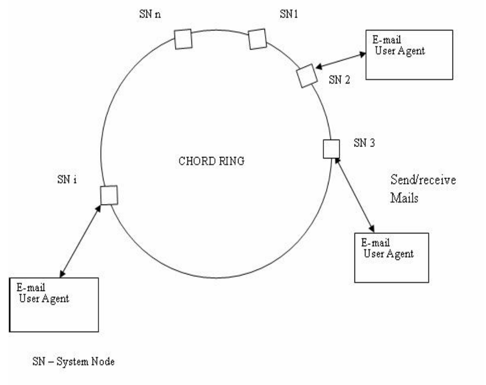
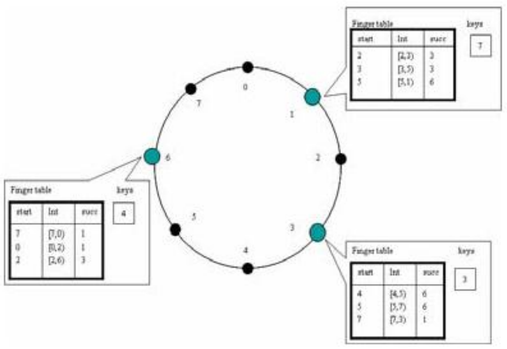


Fig. 1 : Overview of Architecture

Each user has unique email address used for mailing purpose. Generally, the e-mail address certificates bind the e-mail addresses to public keys. These e-mail address certificates are created using external services provided by a certificate authority. A users' public key is contained in the certificate whereas his private key is stored on his local computer in a trusted manner. Each user has an inbox associated to e-mail address, which stores the notification of all unread messages. The messages are not stored in the inbox but are stored separately along with their identifiers. Once the user reads a new message he can store them on his local computer. This system is similar to traditional POP-based e-mail. The difference is in the storage of e-mails. In this architecture the message is stored at k closet nodes of the message. The persistence of the message is guaranteed until the user reads the message. Once a message is read it will be deleted from the peers. Thus, this P2P e-mail system uses the Chord substrate to store three types of objects: e-mail address certificates, unread email message bodies, and inboxes. Each object has a unique identifier. All UAs and system nodes use the same hash function to map an object's identifier to a key, which is an element in the chord ring.

The Chord protocol supports the mapping of given key onto a node. Each node in the chord has a unique ID. Chord ID is assigned an m-bit identifier using consistent hashing. The SHA-1 algorithm [2] is the base hashing function for consistent hashing. Chord is completely decentralized and symmetric, and can find data using only *log (N)* messages, where N is the number of nodes in the system. Chord has a ring-based topology where each node stores at most *log* (*N*) entries (or state) in its *finger table* to point to other peers. Lookup is done in *O* (*log* (*N*)) time.

Each node has a successor and a predecessor located in the next positions clockwise and counter-clockwise respectively. Pointers are stored in every node to predecessor and successor along with pointers to other few nodes. These other nodes provide access in case of node failures of successor or predecessor. An example Chord network with three bit identifiers is shown in Figure 2.



start	Int	succ
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

start	Int	succ
7	[7,0)	1
0	[0,2)	1
2	[2,6)	3

start	Int	succ
4	[4,5)	6
5	[5,7)	6
7	[7,3)	1

Fig. 2: Finger tables and key locations for a ring with nodes 1, 3, and 6 and keys 3, 4, and 7.

Chord is expected to perform three tasks when a node joins the network with at least one node. Those tasks are presented in the form of pseudo-code in [4].

3. Design

Details of Data design and Procedural design are presented below.

A. Data Design

The system node of the chord ring is expected to maintain the following data structures for providing the lookup service.

Finger Table: Finger table entry includes both the Chord identifier and the IP address of the relevant node, the successor and predecessor nodes ids and IP addresses, and start node and end node for the given index.

Node: Node structure contains the node identifier and node IP address.

The E-mail User Agent module is expected to maintain the following data structures for sending and receiving the mails.

Store users: used for storing user e-mail address along with certificate.

Store messages: the message ID and its message body of the each message sent by the user and list of recipients for the message ID.

Inbox: The inbox stores an e-mail address and the encrypted email message headers.

B. Procedural Design

A simple store-and- forward is considered as the basis for the procedural design of the proposed system. A **Store** function is used to store e-mail message body and the corresponding e-mail address certificates. **Delete** function is used by a requester to reclaim storage space in an individual node by removing unneeded objects from its storage. **Fetch** function is used to retrieve stored objects. **Append-inbox** function is used by a sender to append email message headers to a recipient inbox, which is a container of message notifications for the recipient. A UA calls **Read-inbox** function on a node when it wants to retrieve message notifications placed into its user's inbox.

Peer-to-Peer E-Mail System Procedures: As the network comprises of semi reliable nodes the E-mail message data needs to be replicated in a sufficient number of nodes to guarantee persistence. The user agents handle replication of data through the services that are available in the system nodes. The two major tasks are composing and checking mails. These tasks require store, fetch, delete, append inbox and read inbox services provided by each system node in the Chord ring.

Following are the sequence of events that occur when Alice sends an email message to Bob and when Bob wants to read his new message.

Composing E-mail: Following are the sequence of events that occur when Alice sends an email message to Bob [5,6,11]. Let A be Alice's user agent.

1. *A* appends Bob's email address with "-certificate," and maps this to a key (related to email address or email address

identifier). *A* uses the lookup service to obtain the list of *k* nodes closest to this key, and fetches Bob's certificate from one of these nodes. *A* authenticates the certificate, and extracts Bob's public key.
2. *A* generates a session key, and uses it to encrypt the e-mail message body.
3. *A* generates an RFC 822 message ID that will be used to identify the message body.
4. *A* maps the message ID to a key. *A* uses the lookup service to obtain the *k* nodes closest to the key, and invokes the store operation on each of them, using the message ID as identifier, and the encrypted message body as object.
5. *A* constructs the e-mail message headers (which include the session key, the message ID header, and a digest of the message), and encrypts them with Bob's public key. *A* maps Bob's e-mail address to a key. *A* obtains from the lookup service the *k* nodes closest to the key, and invokes the append-inbox function on each of them with Bob's e-mail address and the encrypted message headers.
The figure 3 shows the sequence of steps involved in the composing E-mail.

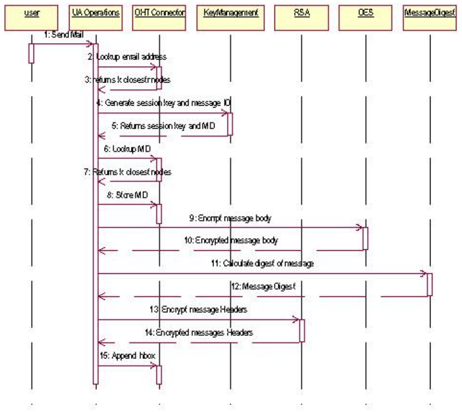

user
UA Operations
DHT Connector
KeyManagement
RSA
DES
MessageDigest
1: Send Mail
2: Lookup email address
3: returns k closest nodes
4: Generate session key and message ID
5: Returns session key and MID
6: Lookup MID
7: Returns k closest nodes
8: Store MID
9: Encrypt message body
10: Encrypted message body
11: Calculate digest of message
12: Message Digest
13: Encrypt message Headers
14: Encrypted messages Headers
15: Append Inbox

Fig. 3 : Sequence Diagram for Composing E-Mail

Checking E-Mail: Following are the sequence of events that occur when Bob wants to read his new message. Let B be Bob's user agent.
1. User agent B obtains the *k* nodes closest to Bob's e-mail address, which are the nodes to which senders are appending message notifications. The inboxes stored across these nodes may be inconsistent. Hence B invokes the read-inbox operation on all *k* nodes, and forms the *superset* of message notifications returned from all *k* nodes. Each node will delete the message notifications once it has sent them to the user agent, and so the user agent becomes wholly responsible for maintaining the persistence of these e-mail message notifications.
2. B decrypts the message headers using Bob's private key. B gets the message ID from the message header. The user agent obtains the *k* nodes closest to the key of the message ID
3. B invokes the fetch operation on one of the nodes, and verifies that the message body is valid by comparing a digest of the retrieved object with a digest that the sender placed in the message headers.
4. If the message body is not on the node, or if the message digest is not valid, B invokes the fetch operation on one of the other nodes, and repeats until it obtains a satisfactory result. Once the message body object has been obtained, B decrypts the object with the session key that the sender placed in the message headers.
5. B invokes the delete operation on all *k* peers to remove its certificate from the list of certificates attached to the message body. If Bob's certificate is the last on the list, the node will remove the object from storage. Because the message headers have left peer storage, there is little motivation for user agents to leave message bodies in peer storage, other than to avoid the consumption of local storage. For this reason, user agents are expected to perform message deletion immediately following message retrieval. The figure 4 shows the sequence of steps involved in the checking E-mail.

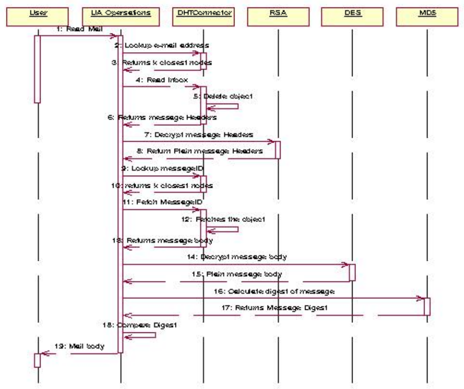


Fig. 4 : Sequence Diagram for Checking E-Mail

4. IMPLEMENTATION

The Implementation comprises system nodes (Chord Substrate) and User Agents (UAs). C++ classes used and the implementation details of various functions in those classes are described below.

A . Chord Substrate

The figure 5 shows the details of classes used and relationships between them in implementing the Chord substrate.

Class End Point: This class acts as an endpoint for the outside module. UdpInitialization method is used for initializing the UDP socket listening point with given port number for sending and receiving the messages. WaitForRequests waits for requests. It uses select system call for selecting the request that came from any outside module. ProcessTheRequests method calls the appropriate function depending upon the request.

Class Node: This class is used for maintaining the finger entries of chord node. getID method is used for generating the unique identifier for each node going to join in chord ring. A node's identifier is chosen by hashing the combination of node's IP address and port number. *lookup* method supports the operation of mapping given a key to a node. Hence this is may be viewed as the Chord Application Program Interface (API) for the outside module. UpdateFingerTable updates the existing finger table of node n. Node n will become the i^{th} finger of node p if and only if p precedes n by at least 2^{i-1}, and the i^{th} finger of node p succeeds n.

Class EmailSystem: This class is used for providing the different services for P2P E-mail system. storeAddress stores the E-mail address and certificate for authentication when a user is publishing E-mail address. readInbox takes an e-mail address and returns the message notifications stored in the node

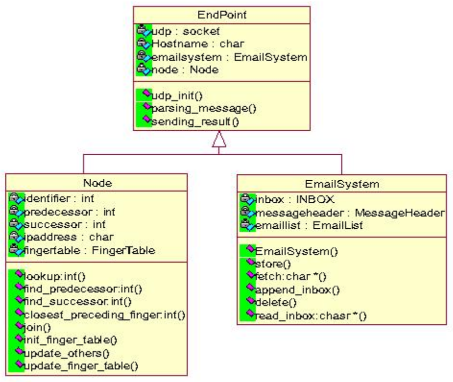


Fig . 5 : Class collaboration Diagram for Chord substrate

B. E-Mail User Agent

The Figure 6 shows the details of classes used and relationships between them in implementing the E-mail User Agent.

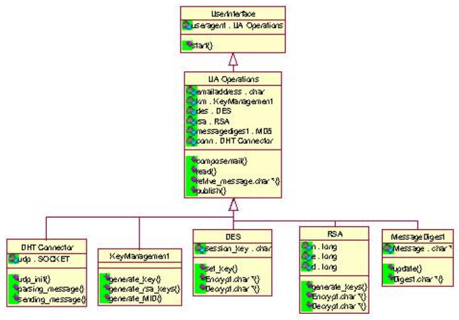


Fig. 6 : Class collaboration Diagram for E-mail User Agent

5.CONCLUSIONS

In this paper we have proposed a P2P E-mail system using Chord as the DHT substrate. The system has been implemented in C++ and tested for functionality at present. At a later stage we wish to identify the generic classes in this work as patterns or a P2P communication framework. Such framework could be easily adopted for various types of mailing systems.

A P2P architecture could be an ideal vehicle for bringing the different forms of communications together. With its distributed advantages and resilient aspects, P2P architecture could be reformed as a P2P framework. This framework can be customized to design and implement many distributed applications of either Internet or other large-scale distributed tasks. Few such envisaged applications are co-operating file sharing, E-mail, time-shared available storage systems, and secure distributed databases.

REFERENCES

[1]. Stoica, I., Morris, R., Karger, D.,Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT LCS, March 2001.

[2]. FIPS 180-1. *Secure Hash Standard.* .S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[3]. Karger, D., Lehman, E., Leighton, F., Levine, M., Levine, D., and Panigrahy, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, TX, May 1997), pp. 654–663.

[4]. R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

[5] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM 21 (1978).

[6] www.rsasecurity.com

[7]. S.Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In proceedings of SIGCOMM, San Diego, CA, Aug.27-31,2001.

[8]A.Rowstron and P.Druschel. Pastry: Scalable, distributes object location and routing for large-scale perr-to-peer systems. In IFIP/ACM International Conference on Distributed Sysytems Platforms, Heidelberg,Germany, Nov.2001.

[9] B.Y. Zhao, J.D. Kubiatowicz, and A.D.Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Technical Report UCB//CSD-01-1141, University of California, Berkeley, Apr.200.

[10]. Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu – *"Peer-to-Peer Computing".* HP laboratories, July 2003.

[11]. www.aci.net/kalliste/des.htm