# MMS Mail System

## CIS 505 Project, Spring 2007

**Debajit Adhikary** (debajit@seas.upenn.edu)
**Khader Naziruddin** (khadera@seas.upenn.edu)
**Ayesha Muntimadugu** (ayesham2@seas.upenn.edu)

April 19, 2007

# 1 Design

## MMS Mail: A Distributed Client-Server Email System

### Abstract

The project builds an implementation of a client-server email system called an *MMS Mail System* comprising multiple servers and clients. The system comprises the following entities:

- **Clients.** Each client initially connects to one of the mail servers and opens a new account. Thereafter, it may connect to any other server to send and receive mail.
- **Servers.** Mail account information is stored across various servers in a distributed manner. Each server can support multiple clients at the same time, and is replicated once for fault tolerance. Each server is aware of other servers in the system.
- **Protocols.** SMTP and POP3 are respectively used to send and receive email. The servers use a synchronization protocol (between themselves) that is transparent to the clients.

### Overall Design

As noted above, the system comprises clients, distributed servers, and the various protocols they communicate with. A broad design of the system is outlined below. System components are detailed later.

- **Distributed Lookup Tables.** Each server has a local copy of two server lookup tables:

Given a username, these are used to lookup the server which contains account and mailbox information for that username. Whenever a new user is created at a server, the server broadcasts a message with that username and its own information to all the other servers, which on receipt, update their local copies of these tables.

| SERVER LOOKUP TABLE | | |
|---|---|---|
| Username | Primary Server | Port |

| BACKUP SERVER LOOKUP TABLE | | |
|---|---|---|
| Username | Backup Server | Port |

This scheme was chosen as it provides the following advantages:

- **No single point of failure**, as opposed to a centralized lookup server.
- **Lookups are cheap** because each server reads its own local copy of the lookup table.
- **Local validations are possible to check if a username exists**. When a user attempts to log in to a mail server, the server can first locally check if the user exists without querying another server, thus reducing network traffic and possibly increasing response time. (If a username exists, only then would the server send an authorization request to the server that contains information on that user).

- SERVER CONFIGURATION.

Information about user accounts and mailboxes is stored in a database (each server has its own). A server configuration file is used to specify the various parameters for database connection. This design also allows multiple servers to share one database host, if required.

| SERVER CONFIGURATION FILE |
|---|
| DB Username |
| DB Password |
| DB Hostname |
| DB Name |

- DATA SYNCHRONIZATION & REPLICATION FOR FAULT-TOLERANCE.

Data related to user accounts and mailboxes stored at one server is mirrored to a backup server. Data is replicated each time any account operation or mail operation (like sending a mail, deleting a mail, creating a folder or creating an account) takes place.

Each time a server updates its database, an **RPC** (remote procedure call) is made to its backup server via a **database trigger**, and the backup server then automatically updates itself with the new information. The backup server then sends an *ACK* (acknowledgement) to the primary server.

In the eventuality that the primary server does not receive an *ACK* from its backup, it adds the update request to a *sync-queue* and periodically attempts to sync its database with the backup.

This design allows:

- **MINIMUM NETWORK TRAFFIC.** Network traffic for backups is minimized because a message is sent to the backup server only when required. This is as opposed to a polling scheme or a scheduled synchronization system, which circulate more messages on the network. (A primary server will reattempt synchronization only if the backup fails to acknowledge the sync request or vice versa)
- **BETTER DATA INTEGRITY.** Consider a scenario where the primary server fails after the user has performed some mail operation. If the user connects again, he would connect to the backup server (since the primary server is down). With this design, the chances of data integrity at the backup server side are maximized because the database trigger would ensure that if the primary database was updated, the backup data would also be updated, thus maximizing data integrity.
- **REDUCED SYSTEM LATENCY.** Since synchronization is carried out immediately after a primary server updates itself, therefore the overall latency of the system is reduced.

  This protocol is discussed in greater depth in the Synchronization Protocol section.

## ASSUMPTIONS

The following assumptions have been made:

- Mail servers cannot be dynamically added or removed from the mail system network.
- Both the primary server and its backup server should not fail at the same time.

# 2 Interaction Diagrams

## USER AUTHENTICATION AND MAIL TRANSACTIONS

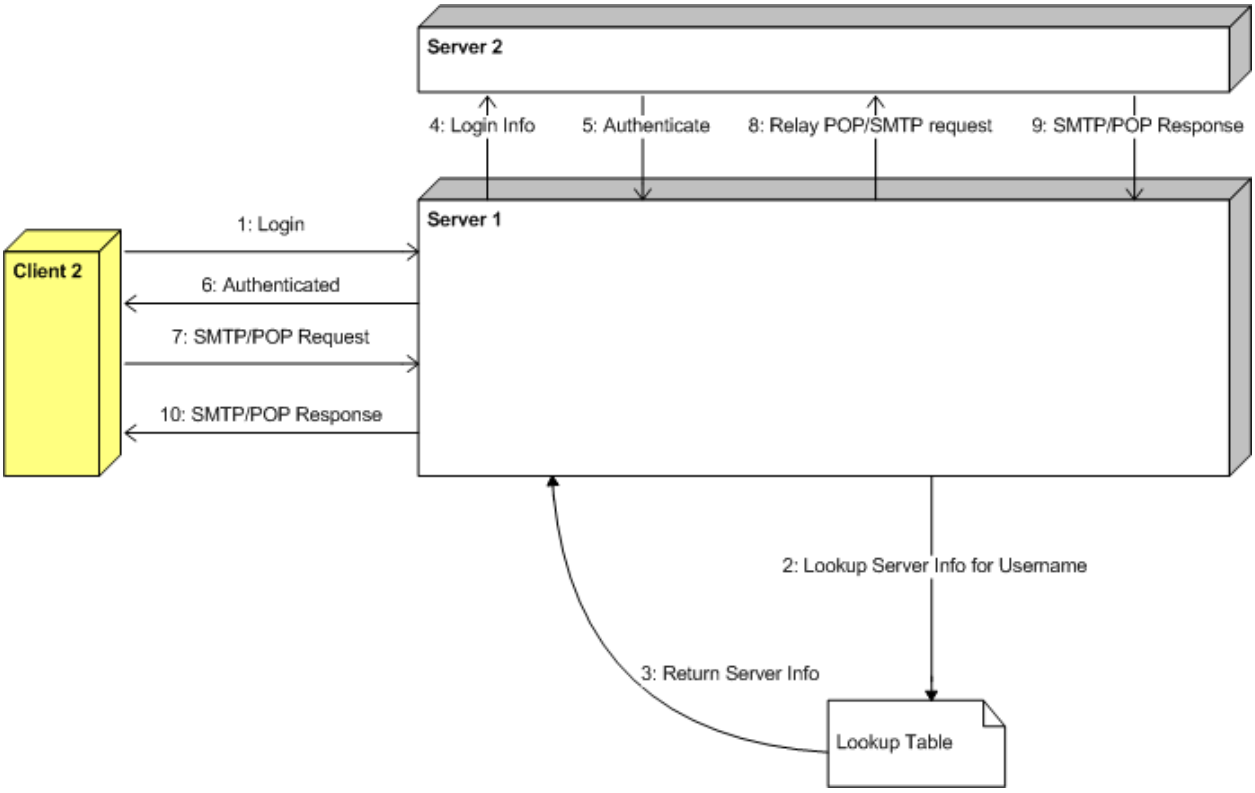


Figure 1: **User Authentication and Mail Transactions**

(In these collaboration diagrams, numbering denotes the relative ordering of a sequence of events and/or messages.)

## NEW USER ACCOUNT CREATION

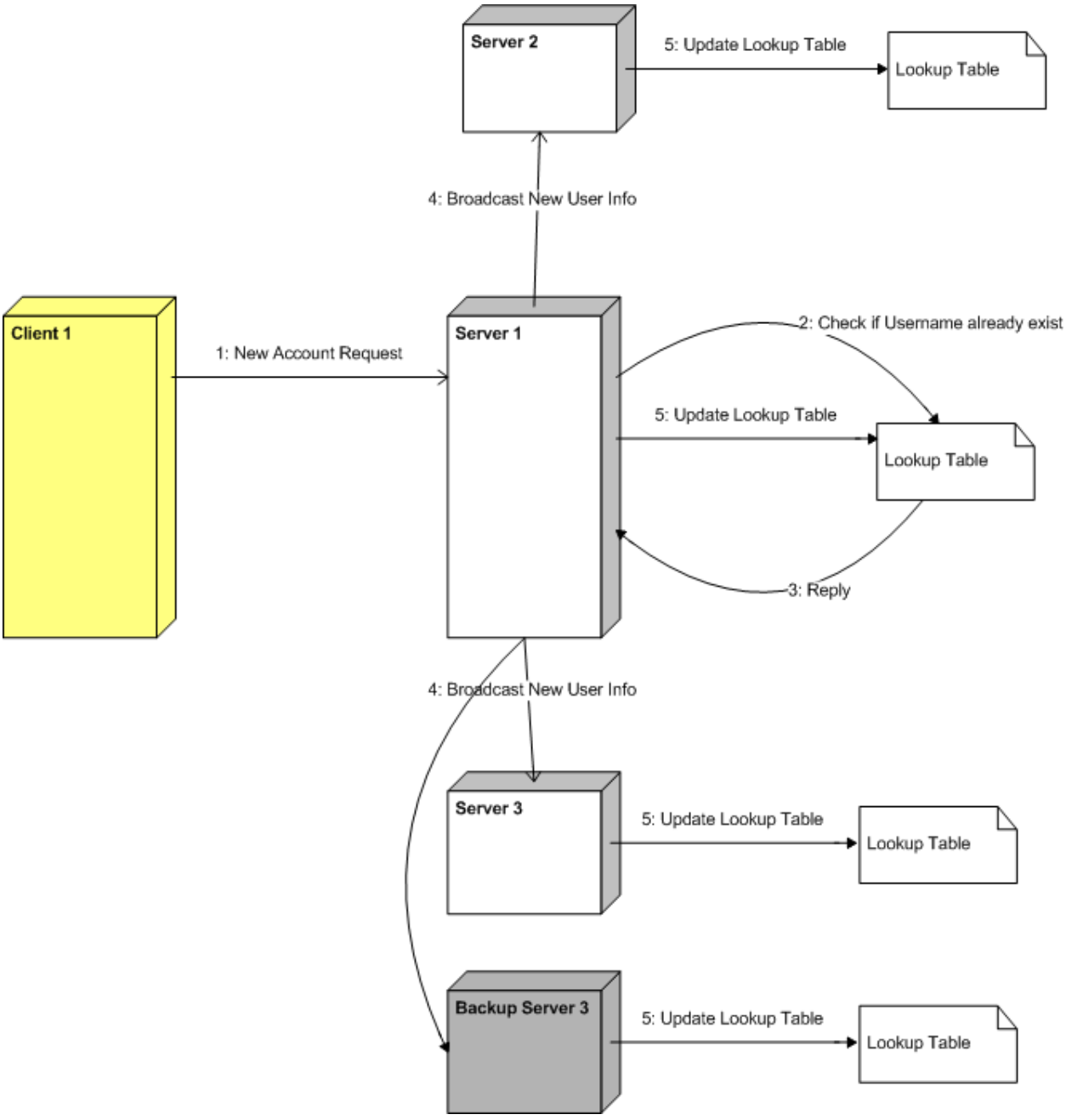


Figure 2: **New User Account Creation**

## BACKUP SERVER SYNCHRONIZATION

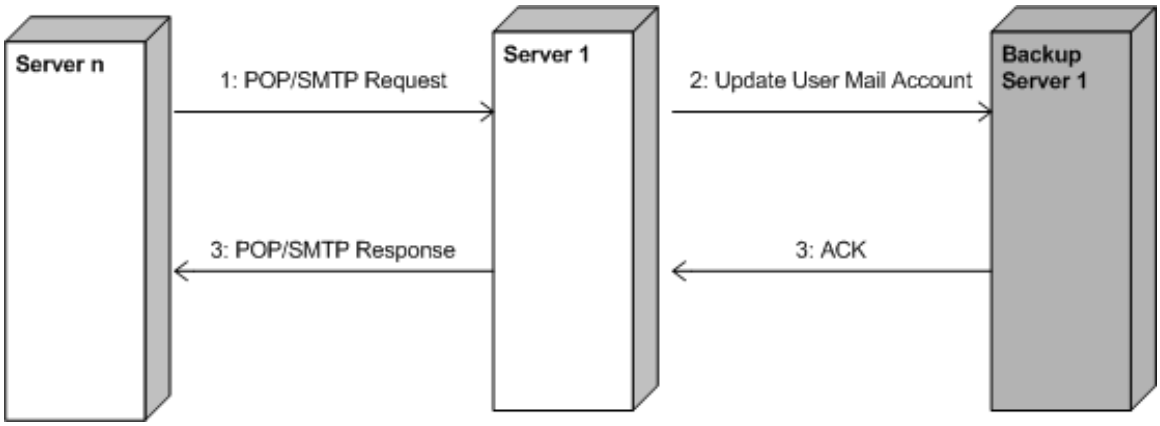


Figure 3: **Primary Server–Backup Server Synchronization**

## PRIMARY SERVER FAILURE

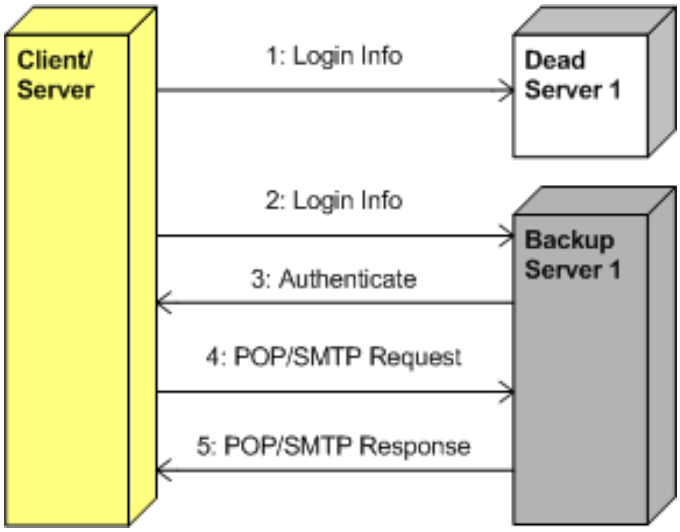


Figure 4: **Interaction in the event of primary server failure**

# 3 Synchronization Protocol

## CSP MODEL

```
PrimaryServer :: Node ? Request ;
                 (Request == Update) ->  UpdateMyself ();
                 Queue ! add(Request);
                 BackupServer ! Request ;
                 PeriodicTimeOut -> [
                                        Queue ! isNotEmpty()
                                        -> * [
                                                 Queue ? Request ;
                                                 BackupServer ! Request
                                                 BackupServer ? Ack;
                                                 AckNotReceived ->
                                                 NewQueue ! add(Request)
                                              ];
                                        setQueue(NewQueue);
                                    ]
```

```
BackupServer :: Node ? Request;
                NodeIsNotMyPrimaryServer -> PrimaryServer;
                NodeIsMyPrimaryServer    -> [
                                               UpdateMyself();
                                               PrimaryServer ! Ack;
                                            ]
```

## ADVANTAGES OF THE PROTOCOL

- Synchronization is guaranteed even if either the backup or the primary is down. (The server that is alive will retry at periodic intervals and will synchronize with its target server (backup or primary) when it comes up again).
- The synchronization protocol applies equally to both the primary and backup server. (If the primary dies, the backup can function as an immediate transparent replacement for the primary and can even synchronize it when it comes back up).
- Only one message is required for synchronization if the backup server is up at that time. (The backup server will then send one acknowledgement message.) This minimizes network traffic.
- Some more advantages have been described in the Data Synchronization section.

**Note:** Since the synchronization protocol is "symmetric" vis-a-vis the primary and the backup server, the terms *primary server* and *backup server* are interchangeable for synchronization issues.

### DISADVANTAGES OF THE PROTOCOL

- If the backup server is down at the time of sending a sync-request, the primary server will reattempt synchronization by sending periodic messages till the backup server comes up again. This will lead to a small increase in network traffic between the primary and the backup.

## 4 Implementation

| | |
|---|---|
| *Language:* | Java (JDK 1.5, Servlets 2.4) |
| *Database Backend:* | MySQL 5.0 |
| *Application/Web Server:* | Apache Tomcat 5.5 (For client user interface) |
| *Mail Protocols:* | SMTP, POP3 |
| *Replication Scheme:* | Data Mirroring |

## 5 Database Schema

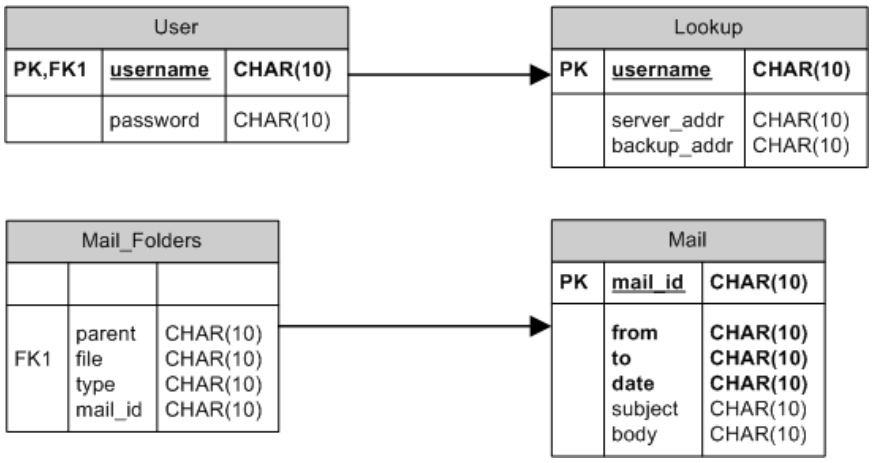


Figure 5: **Database Schema**