# Peer-to-Peer Support for Distributed Mail Transfer Mechanism

Kai-Hsiang Yang, Jenq-Haur Wang, Chi-Chien Pan, and Tzao-Lin Lee
Department of Computer Science and Information Engineering,
National Taiwan University, No. 1, Sec. 4, Roosevelt Rd., Taipei, 106, Taiwan.
E-mail: {f6526004, jhwang, d5526001, tl_lee}@csie.ntu.edu.tw

**Keywords: peer-to-peer technology, location service, load balancing, personalization, mobile IP**

## Abstract

**In current Internet mail transfer mechanism, mail servers usually do a lot of extra mail processing like mail filtering. The workload of mail servers is heavy in terms of storage and processing time. Junk mails as well as important messages are all stored on mail server, retrieved and then deleted by end users. It's a waste of processing time, storage space, and precious network resources.**

**In this paper, peer-to-peer technology was included in ordinary mail transfer mechanism to reduce unnecessary overhead. Through the user location service, mail server can dynamically query the online status and current location of users. By redirecting mails to online users' hosts, workload of mail servers will be greatly reduced, and personalization of mail processing configuration can be better supported.**

## I. INTRODUCTION

With the tremendous growth of the Internet, various networking applications such as WWW (World-Wide Web), E-mail, and FTP (File Transfer Protocol), have been widely used. Specifically, e-mail applications have become indispensable and critical to many people's daily lives. All kinds of information like important messages, notifications, and advertisements, to name just a few examples, arrive at your mailbox by e-mail without classification. However, in current mail transfer mechanism, mail server plays a crucial role since every step in mail delivery requires the intervention of mail servers. The protocol used in mail delivery, Simple Mail Transfer Protocol (SMTP) [1], is store-and-forward in nature where guarantee of mail delivery is the main concern. Therefore, mails in the process of delivery will have to be queued in every intermediate mail transfer agent (MTA) before arriving at the destination mail server. This takes too much overhead in network bandwidth and processing time.

When a user gets online, a mail user agent (MUA), such as Outlook Express or Netscape Mail, can be used to check if there are new e-mails or not. Mails are retrieved from a mail server via POP3 (Post Office Protocol version 3) [2]. However, both junk mails and important messages arrive at the same mail server regardless of their priority. Users can only retrieve each mail in order, filter them automatically or manually, and then delete unwanted e-mails. In this architecture, mail server plays a very important role, and wastes its storage and processing time.

There is one well-known filtering language, SIEVE, [17] for users to write some mail filters in the mail server side or in the user’s side. One current mail reader “Mulberry” can generate both kinds of filters. These filters could filter mail at time of final delivery, when the message is moved to the user-accessible mailbox. However the mail servers still have to deal with these junk mails, save them into the user-accessible mailbox, and then run these filters to filter incoming mails. Mail servers don’t decrease its load but increase its load for filtering mails.

Another protocol called IMAP4 (Internet Message Access Protocol version 4) [3] provides more advanced mail management functions, for example, one can get mail headers first before deciding which e-mails to get. Mail synchronization problems can also be easily resolved. However, not all mail servers implement IMAP4 functions. Most users still have to use POP3 clients for mail retrieving. But mail servers using IMAP4 still have to deal with all incoming mails and waste their resources.

On the other hand, peer-to-peer technology has been widely deployed in various applications, for example, file sharing applications like Napster and ezPeer, instant messaging software like ICQ and MSN Messenger, and open source protocols like Jabber [4] and GnuTella [5]. Moreover, decentralized systems evolve towards centralization if scalability is concerned, and centralized applications evolve towards decentralization [6]. We have to find out what combination of centralization and decentralization works best for current Internet applications. Therefore, an infrastructure for integrating current Internet mail transfer mechanism and peer-to-peer instant messaging was proposed to provide better services.

## II. MOTIVATION

We will quickly review the current architecture for mail transfer and its shortcomings, and then propose our infrastructure as a feasible solution.

In current Internet mail transfer mechanism, separate protocols are used: SMTP [1] for mail delivery and POP3 [2] or IMAP4 [3] for mail retrieval and management. As shown in Fig. 1, a typical scenario for current mail transfer mechanism is illustrated.

As shown in Fig. 1, a user composes his e-mail by a MUA like Outlook Express or Netscape Mail and then sends it to the MTA in his domain (or the ISP he connects to), or sender MTA, via SMTP (step 1, 2). Sender MTA then checks the recipient e-mail address for its validity and determines how to relay this mail to the right person. Usually the MX Record of the DNS (Domain Name System) [7] server will be queried for the mail exchanger of the domain specified in the recipient e-mail address (step 3, 4). After a relayed mail is received, (step 5, 6), receiver MTA will store it into the recipient mailbox. When the recipient gets online, he can use a MUA to check and retrieve his own emails from the mail server via POP3 or IMAP4 (step 7, 8). No matter if the recipient is online or not, e-mails will be delivered to his domain mail server first. Then after the user decides to check his e-mails, he will connect to the mail server and fetch them back. Mail servers cannot notify users of incoming mails unless their MUA is configured to periodically check for new e-mails.

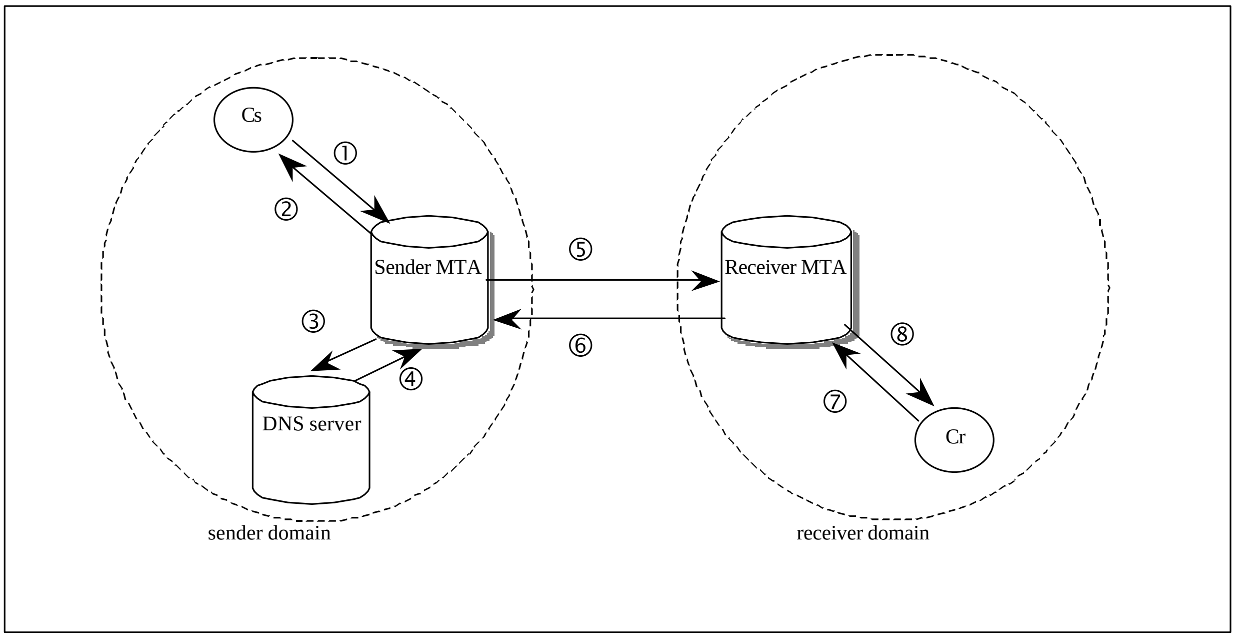


Fig. 1. Typical scenario for mail transfer and retrieval where

Cs: mail sender, Cr: mail recipient,

*Sender domain*: home domain for sender,

*Receiver domain*: home domain for receiver,

*Sender MTA*: MTA for sender domain,

*Receiver MTA*: MTA for receiver domain,

*User-side SMTPd*: SMTP daemon on the user side

This process works fine, but there are several drawbacks that affect the performance of mail servers. Firstly, the workload on a mail server is heavy in terms of storage for mailboxes and processing time for SMTP/POP3/IMAP4. Mail server has to deal with every email destined for domain users no matter they are currently online or not. This could be a waste of server storage and processing time since emails are unnecessarily stored in server and then retrieved by on-line users. As the number of users and emails grow, the storage requirement of mails in receiver MTA will become larger and larger.

Secondly, personalization cannot be done very efficiently in mail server. For example, it's common to configure an anti-spam list on receiver MTA for the whole domain. But for each individual domain user, different configurations may be needed. We need a finer-grain control of such configuration for each individual user, for example, a separate anti-spam list for each user, which is more reasonable since each user may want to filter mails from different senders and hosts. Although it's possible to configure external programs for mail processing, for example: SIEVE [17] or *procmail* [8] or Milter API [8] in *sendmail* [10, 11] version 8.10 or above. But it's still time-consuming for mail server to deal with personal configurations for all domain users.

In current implementations junk mails are removed as soon as possible after a user checks and retrieves his e-mails from server. That would be a waste of time and space for storing these unwanted junk mails in server followed by deleting them anyway.

In order to offload mail server and to provide complete customization in mail processing, a peer-to-peer infrastructure for mail transfer was proposed. Specifically, we want to bypass the mail server if the recipient is currently online and ready for receiving emails. Users can customize their personal configurations for all kinds of mail processing.

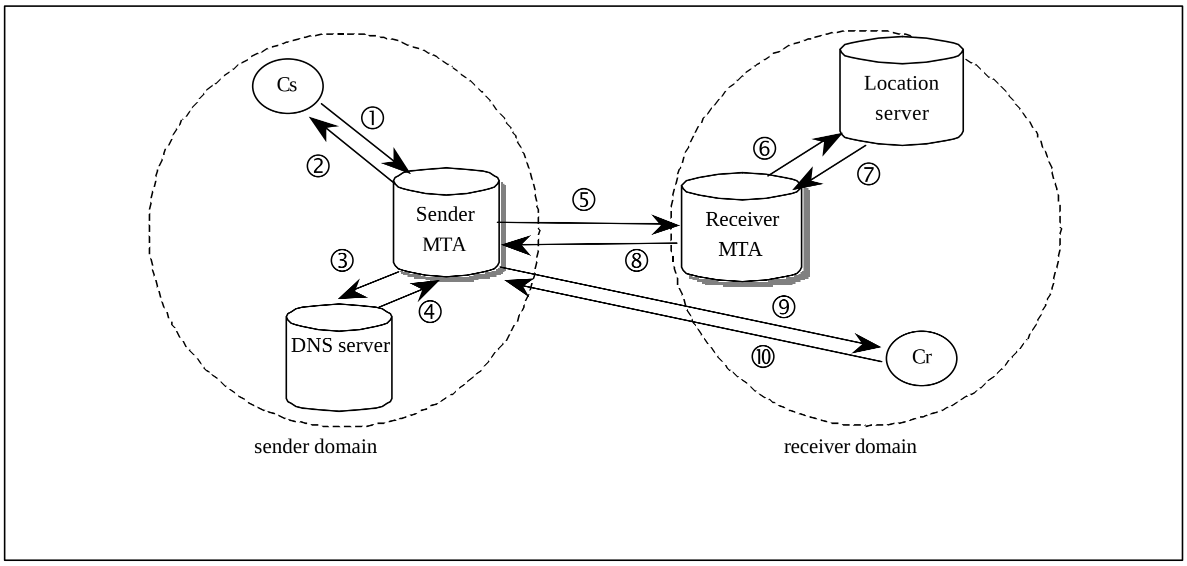


Fig. 2. Our infrastructure for peer-to-peer mail transfer

## III. INFRASTRUCTURE

As shown in Fig. 2, key components in the infrastructure include: location servers, DNS servers, mail servers (sender and receiver MTAs), and mail clients (sender and receiver MUAs). The functional description of each component is provided as follows.

### *A. Location Server*

Location server is responsible for storing the current location and way of contact for each domain user. User Location Record (ULR) includes user name, e-mail address, current online status, current IP address, device capabilities (audio/video support), and user profiles like access control list (ACL) for user resources. Since the amount of data may be quite large, a distributed scheme may be used, for example, one location server for each domain (like DNS server) may be a feasible way. ULR of each user is stored in the location server of his own domain as specified in the e-mail address.

Most of the related works in location service are about geographical positioning of mobile nodes in a wireless network, the location of servers, and location-based services. They mainly focused on the physical positioning of mobile nodes or server, not the current way of contact for clients and users.

As shown in Fig. 3, there are two possible operations for a location server: *update* and *query*. Clients *update* the ULRs to location server when users login, logout, change their location, or modify their configurations. On the other hand, mail servers *query* the location server for ULR of a particular user in order to directly deliver e-mails to him. In other words, location server has to be coupled with the management of user sign-on and sign-off. Users must do registration/de-registration when sign-on/sign-off.

However, when mobile user is roaming into a foreign network, he must register to his home location server for location update. This can be done directly or through the help of foreign location server (*Indirect Update*). For a mobile node to detect its movement into a foreign network, the mechanism of *Agent Advertisement/Solicitation* in Mobile IP [12] can be deployed.

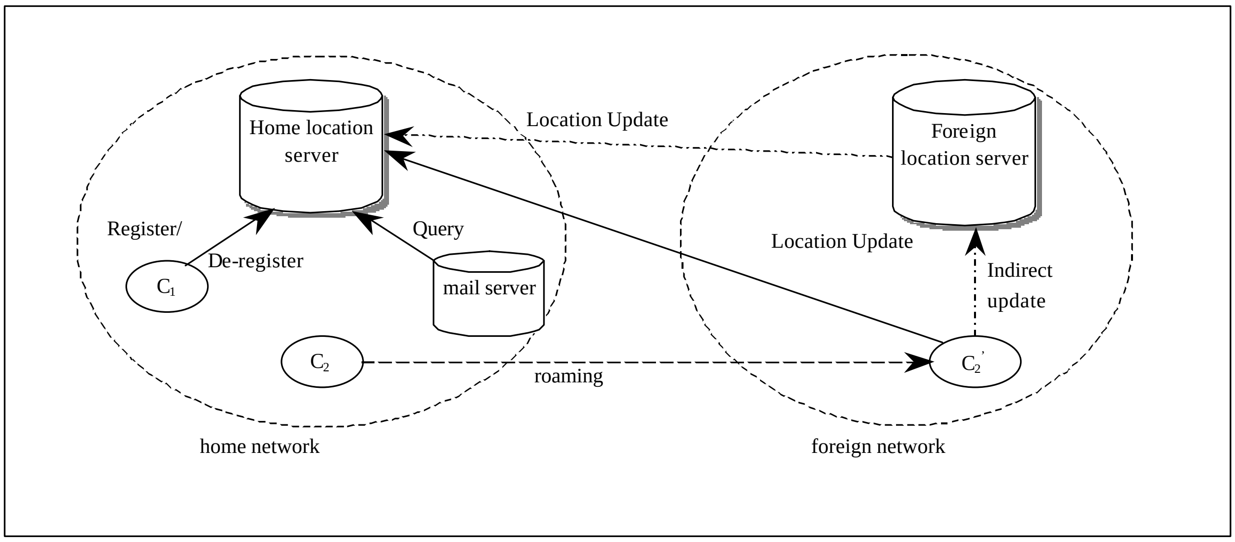


Fig. 3. Operations of location servers.

## B. *Mail Transport Agent (MTA)*

As shown in Fig. 2, after sender forwards his mail to his mail server (sender MTA) (step 1, 2), sender

MTA will query the MX Record of DNS server for mail exchanger (step 3, 4) in order to know which mail server to redirect to. When connected from sender MTA (step 5), receiver MTA will not receive the e-mail directly. Instead, location server is queried for the online status of recipient (step 6). If he is not on-line, mail gets delivered in its normal way, and saved in recipient mailbox.

On the other hand, if recipient is currently online and ready for receiving mail (step 7), receiver MTA will reply to sender MTA a *SMTP REDIRECT* message (SMTP return code 251/551 [1]) (step 8), and sender MTA will send mails directly to recipient host. Then user-side SMTPd will check the validity of destination e-mail address and start receiving his e-mail (step 9, 10). After reception of e-mails, MUAs will be popped up for recipient to read e-mails.

When the recipient is online, he can choose his online status to: 1. ready for receiving mail and 2. busy but notification message only. If the online status is 1, the mail server could redirect to the recipient's host. But if the online status is 2, the domain mail server should receive this mail and notify the recipient by sending a short message via SMTP to recipient's host. Then the SMTPd pops up one message box to notify the recipient of the incoming mail.

One possible error may occur under the circumstances when the recipient host cannot be reached by sender MTA. Possible reasons could be abnormal broken connection or power failure that may not be immediately reflected in location server. In such cases, sender MTA will queue this mail delivery request in its local waiting queue as usual. After a configurable timeout, sender MTA will retry transmission to the normal mail exchanger (receiver MTA) queried from MX record of DNS server, not directly to user-side SMTPd since user location may have been updated again. This is very important for the implementation issue.

There is one design issue for location server. The user on-line status stored in a location server could be inconsistent with his exact location due to possible reasons as power failure or broken connection. We didn't focus on maintaining the timeliness of location information in a location server. Alternatively, applications making use of location information are the ones that try their best to query location server just before connections are to be established to the user host. The reason for such design is user host mobility can be checked only when needed since it may often change.

One design alternative is to modify mail servers to query ULR from a location server, instead of querying DNS server for MX record. But every application must be modified. Therefore, another alternative is to modify DNS server for handling user location queries. ULRs can be stored in a DNS server, and DNS protocol has to be modified since the input of ULR queries is user e-mail address, not hostname. Thus every application using DNS service will be able to utilize the user location service.

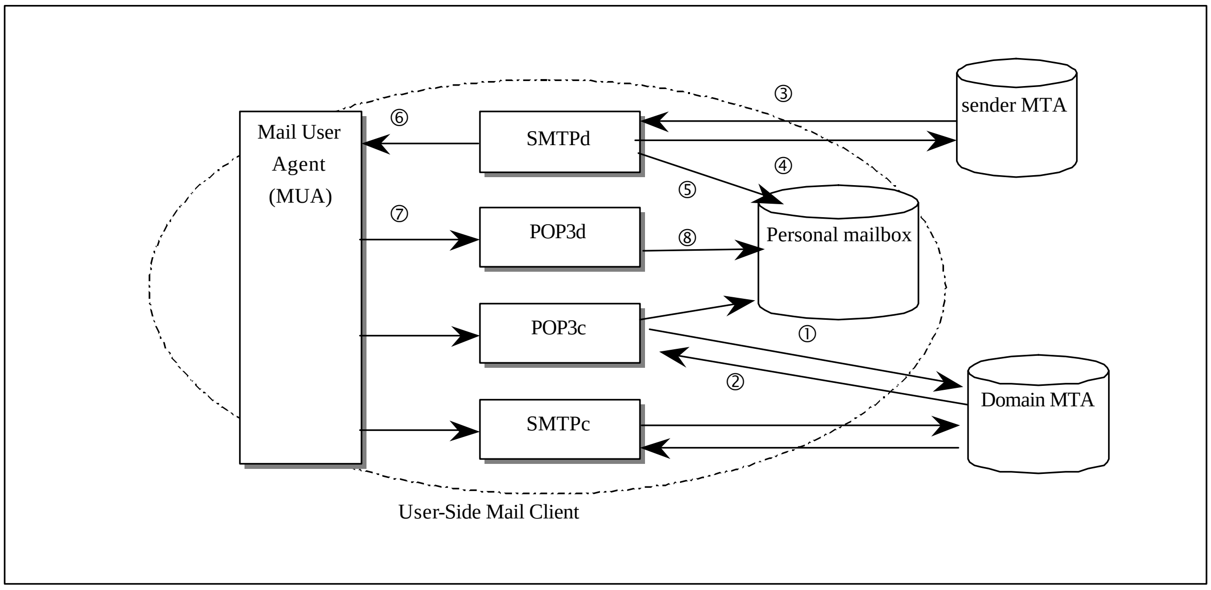


Fig. 4. Operations of a user-side mail client

### *C. Mail User Agent (MUA)*

As shown above in Fig. 4, SMTPd and POP3d (POP3 daemon) are needed in addition to the original SMTPc (SMTP client) and POP3c (POP3 client) in order to receive e-mails from sender MTA and to automatically pop up MUA for notifying recipient. Specifically, when user logs in, POP3c should be configured to automatically receive mails from domain mail server via POP3 (step 1, 2), and then SMTPd and POP3d will be initiated waiting for incoming connections from sender MTA. After mails are received (step 3, 4) and saved in a personal mailbox (step 5), MUA will be triggered (step 6) and popped up for receiving mails from its default POP3 server (step 7), the local POP3d, which in turn reads the saved mail in personal mailbox (step 8).

Note that mails delivered to an off-line user will be stored in mailboxes on receiver MTA. When recipient gets online, POP3c automatically receives mails from domain mail server in order to keep the original mail order in the local personal mailbox.

## IV. ADVANTAGES

In this infrastructure, several advantages are possible. Firstly, load balancing between mail server and clients can be achieved. Mail servers can be offloaded since they don't have to receive e-mails when recipient is online.

Secondly, finer-grain personal configuration for mail processing like mail filtering can be done separately on each client in a distributed way, which further contributes to more offloading from mail servers. Global mail filtering can still be done on mail server.

Thirdly, peer-to-peer support can be integrated into current mail transfer mechanism. Users will be automatically notified of their new mails immediately when they get on-line.

Lastly, mail clients with different levels of capabilities, such as the presence of POP3d/SMTPd functionality, can be integrated in this infrastructure since capability information is also available through our location server. ULR query also serves as a way of capability exchange.

## V. IMPLEMENTATION ISSUES

In this section, we introduce our implementation method.

### A. *Location Server*

Since the design of LDAP (Lightweight Directory Access Protocol v3) [13] server is optimized for reading, and the tree structure in LDAP is easy to maintain the different level of user information, we choose LDAP server as the location server. In our experiment, we use the “OpenLDAP” system developed by LDAP community which is an open source [14]. Besides, we also use the DB library developed by the University of Berkeley.

To store the user location information, we design one new object “ULR” (user location record) in LDAP server, and also define some suitable attributes as we need:

(1) *Username*: the name to login.

(2) *Userpassword*: the password to login and receive email.

(3) *Email*: the email account.

(4) *Online*: show the user’s online status ( 0: Offline, 1: Ready for receiving mail, 2: Busy but notification message only. )

(5) *Ipaddress*: the IP address of the current user computer.

In practical, we insert the following definitions of attributes and objectclass into the LDAP schema:

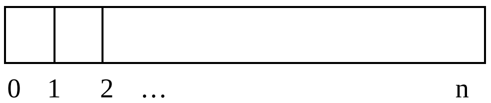
```
(1) attributetype ( 9.8.7.6.5.4.3.2.1 NAME ( 'username' ) SUP name )
(2) attributetype (9.8.7.6.5.4.3.2.2 NAME 'userpassword'
EQUALITY octetStringMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.40{128} )
(3) attributetype (9.8.7.6.5.4.3.2.3 NAME ( 'email' )
EQUALITY caseIgnoreIA5Match
SUBSTR caseIgnoreIA5SubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{256} )
(4) attributetype (9.8.7.6.5.4.3.2.4 NAME 'online'
EQUALITY numericStringMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.36{16} )
(5) attributetype (9.8.7.6.5.4.3.2.5 NAME 'Ipaddress'
DESC 'IP address as a dotted decimal, eg. 192.168.1.1, omitting leading zeros’
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{128} )
(6) objectclass (9.8.7.6.5.4.3.2.6 NAME 'UserLocationRecord'
SUP organizationalPerson STRUCTURAL
MAY ( username $ userpassword $ email $ online $ Ipaddress )
)
```

When one user logins, the user-side software automatically registers to the location server, and then update the Ipaddress and online status. When a user closes the software or logouts, it will automatically cancel the Ipaddress and set the online status to 0.

### B. *Personal Mail Filter*

In our system, each user could set his mail filters to avoid receiving garbage mails. These personal mail filters are stored in the location server, and are read by the login procedure when the user logins. For the mail filter, we design one ordered list of *(operation, right)* pairs. The “*operation*” field contains: (1) <sender e-mail> (2) <sender host> (3) <sender domain> (4) <relay host>. The “*right*” field contains: (1) allow (2) deny.

In our implementation, we use one combined string to show the *(operation, right)* pair as follows:



[Octet 0]: represent the *right* field. ‘1’ is “allow”, and ‘2’ is “deny”.

[Octet 1]: represent the *operation* type. ‘1’ is <sender e-mail>, ‘2’ is <sender host>, ‘3’ is <sender domain>, and ‘4’ is <relay host>.

[Octet 2-n]: the real domain name, IP address, or email.

For example, the following two mail filters make the domain (140.112.4.*) acceptable, but the IP address (140.112.4.11) is unacceptable.

“13140.112.4.*” => (140.112.4.*, allow)

```
"22140.112.4.11"    =>   (140.112.4.11, deny)
```

Against this design, we define one more attribute and add it into ULR object.

```
(7) attributetype (9.8.7.6.5.4.3.2.7 NAME 'mailfilter'
DESC 'Personal mail filter'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{128} )
(8) objectclass (9.8.7.6.5.4.3.2.8 NAME 'UserLocationRecord'
SUP organizationalPerson
STRUCTURAL
MAY ( username $ userpassword $ email $ online $ Ipaddress $ mailfilter)
)
```

One example of "UserLocationRecord" data when user "james" logins is listed as follows:

```
{
dn: cn=james,o=oanet
objectclass: top
objectclass: UserLocationRecord
username: james
userpassword: 1234
email: james@oanet.ntu.edu.tw
online: 1
Ipaddress: 172.16.30.25
mailfilter: 13140.112.4.*
mailfilter: 22140.112.4.11
}
```

## C. *Mail Server*

In our experiment, we choose the *sendmail* [10] package based on Linux environment. We modify *sendmail* to check the user status from location server when new mail is coming. If the online status is 1, it sends the redirect message back to sender mail server for sending the mail directly to the user. If the user is offline, it receives the mail for the user as usual. If the online status is 2, it receives this mail first and then notifies the recipient by sending a short message via SMTP to recipient's host. Then one message box will be popped up to notify the recipient.

### D. *Personal SMTP/POP3 Daemons and Login Procedure*

In the user side, we focus on the Windows platform and develop the login/logout software and SMTP/POP3 daemon by the Java language. When one user logins, he has to input his username and password, then the login procedure connects to location server for authentication, and reads the user's mail filters in his ULR record if it passes the authentication. After these operations finish, the POP3 client immediately connects to domain mail server for receiving mails, and then we run SMTP daemon with parameters *(username, mail filters)*, and run POP3 daemon with parameters *(username, password)*.

## VI. FUTURE WORK

There are some cases where the infrastructure still needs some modifications. Firstly, we have to deal with the cases for users behind firewall or NAT (Network Address Translator) [15]. In the case of firewalls that only allow certain types of traffic (for example, HTTP) to pass through, some encapsulation methods could be used, for example, Firewall Enhancement Protocol (FEP) [16]. For users inside a private network or NAT, some address and port translation has to be done, for example, Network Address Port Translation (NAPT) [15]. Secondly, we can also implement peer-to-peer support for other applications like FTP and HTTP, and a universal messaging service will be possible. Finally, security issues for Instant Messaging (IM) are also likely to occur in such environment, for example, IM Virus for MSN. All these need further considerations.

## VII. CONCLUSION

In the fast-changing world of efficiency, instant messaging and communications are critical for all people. The rapid growth of wireless devices and technology facilitates broader range of applications in wireless communications. Location service plays a major role in such an environment where user mobility management must be maintained for various services.

In this paper, an infrastructure for peer-to-peer mail transfer mechanism was proposed for offloading mail servers and providing better personal customization on mail processing. This infrastructure can also be applied in all kinds of Internet applications where peer-to-peer support is needed.

## REFERENCES

[1] J. Klensin, Ed., "Simple mail transfer protocol," *RFC 2821*, April 2001.

[2] J. Myers and M. Rose, "Post office protocol – version 3," *STD 53*, *RFC 1939*, May 1996.

[3] M. Crispin, “Internet message access protocol – version 4rev1,” *RFC 2060*, December 1996.

[4] Jabber Software Foundation, *Jabber*, available at: http://www.jabber.org/.

[5] *GnuTella*, available at: http://www.gnutella.co.uk/.

[6] A. Oram, “Peer-to-peer for academia,” *available at: http://www.openp2p.com/pub/a/p2p/2001/10/29/oram_speech.html*, October 2001.

[7] P. Mockapetris, “Domain names – implementation and specification,” *STD 13*, *RFC 1053*, November 1987.

[8] S. R. Berg and P. Guenther, *procmail*, available at: http://www.procmail.org/.

[9] Sendmail, Inc., “Filtering mail with sendmail,” *available at: http://www.sendmail.com/partner/resources/development/milter_api/*.

[10] Sendmail Consortium, *sendmail*, available at: http://www.sendmail.org/.

[11] B. Costales and E. Allman, *Sendmail*, *2nd ed.*, O’ Reilly & Associates, Inc., January 1997.

[12] C. Perkins, “IP mobility support for IPv4,” *RFC 3220*, Jan. 2002.

[13] M. Wahl, T. Howes, and S. Kille, “Lightweight directory access protocol (v3),” *RFC 2251*, December 1997.

[14] OpenLDAP Foundation, *OpenLDAP*, available at: http://www.openldap.org/.

[15] P. Srisuresh and K. Egevang, “Traditional IP network address translator (traditional NAT),” *RFC 3022*, January 2001.

[16] M. Gaynor and S. Bradner, “Firewall enhancement protocol,” *RFC 3093*, April 2001.

[17] SIEVE: A Mail Filtering Language. *RFC3028*, January 2001.