Proceedings of FREENIX Track:
2000 USENIX Annual Technical Conference

San Diego, California, USA, June 18–23, 2000

EXTENDING INTERNET SERVICES VIA LDAP

James E. Dutton

Extending Internet Services Via LDAP

James E. Dutton
Southern Illinois University
jimd@siu.edu, http://www.usenix.org/events/usenix2000/freenix/dutton.html/

Abstract

This project report examines the use of an LDAP (Lightweight Directory Access Protocol) V2 server to provide an easily accessible data storage facility. The main purpose of the LDAP database is to store related information based on a common thread such as a person's name, an organization's name, or the description of a service offered, in a simple yet hierarchical structure.

The use of LDAP enables new fields to be added to existing user information to 1) enable end-users to store pertinent user information to be used by a mainframe-to-PC intermediary file server using Samba, 2) enable new groupings of electronic mail distributions to be created with little or no change to Sendmail, and 3) enhance the granularity of InterNetworkNews (Usenet) article submission acceptance capabilities.

Some additional benefits of these facilities included using a single, non-proprietary database which required very little new coding to make use of. The data used for the various facilities were easily associated with database objects defined for enterprise personnel. The administration load for each service was reduced since service related data, such as userids or mailboxes, were not maintained directly as a part of the specific service. The Internet Directory Service, as provided by the LDAP server, is accessible by several methods, rather than just one specialized or proprietary interface.

1 Introduction

With the advent of Internet Directory Services based on the Lightweight Directory Access Protocol (LDAP) standards, it is now easier to enhance, or extend, existing Internet Services as well as create new ones. An Internet Directory Service need not be limited only to user information for the benefit of electronic mail clients.

Three specific Internet Services were enhanced through the use of an LDAP server. They included: e-mail servers using Sendmail, Usenet or Network News servers using InterNetworkNews (INN), and file transfer servers using Samba[14] (PC NetBIOS file system on a UNIX host) and Expect[13] (an interactive process scripting language for UNIX).

As its basic function, the LDAP server provided a data storage facility which was easily accessible using a Perl script or simple modifications to supplied application configuration code (Sendmail). This avoided developing specialized databases with specialized program interfaces. Also, no fundamental changes in each of the Services were required.

The LDAP server also provided a central data-store, much as a traditional database, where many sets of information, such as a list of authorized Network News posters for a given internal or local newsgroup, a list of Internet hostnames associated with some specific services such as electronic mail and Network News, and a list of people within certain organizational groups which may be independent of the other lists, could be kept – regardless of their association with each other. In other words, there is no need for a specialized database just for electronic mail, and another one for Network News, and yet another one for the file transfer mechanism. At the same time, the data used for these enhanced Internet Services could be, and were, related to a specific user or entity. In the particular case of Sendmail, the LDAP server replaced one of the Sendmail user databases.

Another benefit to using an Internet Directory Service was the fact that there are multiple publicly available access methods, such as Frank Richter's Web500GW Web interface[11], Kartik Subbarao's *ldaptool.html*[18] for use with Web browsers, or one of the University of Michigan's specialized, yet freely available, interface programs (i.e., Wax500, Max500, Xax500). This meant that a commercial or proprietary program is not needed to use or access the LDAP data, which means easy maintainability and reduced costs.

At the same time, access to specific data items within the server can be easily restricted without writing specialized access application programs. Using LDAP Access Control Lists (ACLs), the appropriate access permissions are established for some of the data used in these facilities.

While each of the three services enhanced with LDAP could have provided its own facility to store and retrieve some of the user/service data required, this would have led to extra maintenance for each service. Using the Internet Directory Service method, very little maintenance is required to store and obtain the necessary data for the enhanced services. After the installation or addition of the relatively small sets of code for each service to access the LDAP data, all maintenance is then left to just the LDAP service itself.

A late addition to this project is a simple Perl script to formulate an LDAP query specifically looking for information about a LAN Administrator, and then display the result in a simple table. This required a locally installed portion of the LDAP suite (LDAPSEARCH) to perform the actual LDAP work. The Perl script's main function was to take the user's input, generate the LDAPSEARCH parameters, execute the LDAPSEARCH program, and display the selected output in a more user-friendly format.

Another late addition to this project is a simple JavaScript function for a Netscape V4 browser (which has built-in support for LDAP) that implements a simple but effective search for LAN Administrators based on one of several criteria. This is another example of a relatively easy method to extend the use of and access to data stored in an Internet Directory Server.

Lastly, this report does not discuss the implementation of network and data security, including passwords. Some general comments on security, however, can be found in Section 11.

2 Executive Overview of LDAP

The Lightweight Directory Access Protocol is an Internet standard that brings X.500 Directory services to the Internet. Implementations of LDAP provide for a structured database with entries, referred to as "objects," formed by single-word character key and multi-word/-line character value pairs. A few special attributes provide for binary data values. A standard set of keys are well defined in the LDAP standard and can be considered as user friendly, for the most part. In general, the keys are usually abbreviations of an LDAP attribute name. For example, two common keys, or attributes, are *dn*, Distinguished Name (DN), and *cn*, Common Name (CN).

Many of the LDAP attributes are used to store, human readable information about people and organizations. The attributes usually identify something specific about a person or organization such as an electronic mail address (*mail*), or a commonly used name, nickname, or pseudonym for a person, organization, or organizational unit (*cn*), or computer userid (*uid*).

The basic LDAP implementation usually provides for clear text password authentication only. This means that when an LDAP client is required to send a password to an LDAP server, that the password is not encrypted, but is sent as plain or clear text. Some specific implementations or site provided add-on programs may provide for secure client, or user, authentication using Secure Sockets Layer (SSL) or other mechanisms, but this is not yet a function of the LDAPv2 protocol itself. Authentication is the process of sending a user-identifying data string, commonly an LDAP Distinguished Name from an LDAP database object entry, and its associated password string. This is normally required only when updating an LDAP database entry. Most LDAP queries are performed without any authentication, and appear to the LDAP server as a "null" or undefined user or client. User/client authentication is independent of access control mechanisms, ACL lists (see Section 10).

Some descriptions of LDAP liken it to an electronic telephone book, or "yellow pages" directory, though that is only part of what LDAP can be used for. LDAP databases are most often organized in a tree or hierarchical structure. A large structure may be distributed over more than one LDAP server, and may include references to other LDAP servers, providing for a distributed directory service.

3 Why Choose LDAP?

Many database systems are available, so why choose an LDAP database? Without going into great lengths to examine all of the possibilities, here are some reasons for choosing LDAP versus other database or X.500 (DAP) products:

- Two well known and publicly available LDAP packages are LDAP-3.3[6] from the University of Michigan (UMich) and OpenLDAP-1.2.x[15] from the OpenLDAP Project.

- Netscape V3/V4 Directory Server is a good commercial implementation of an enhanced/updated UMich LDAP-3.3, which provides another good LDAP source that closely follows the open (IETF) LDAP standards

- some X.500/DAP servers are not user-extensible, which is not a limitation of X.500/DAP per se, but of the particular implementations. The LDAP implementations mentioned in this paper are all user-extensible. Where a given X.500/DAP server is user-extensible, then it may be used in place of an LDAP server.

- LDAP compliance and compatibility is appearing in more and more user and networking software
- in many cases, a proprietary interface is not needed to access an LDAP database, as is true with at least some commercial database systems and some commercialized X.500/DAP databases which require a specialized access client program that works only with that particular system
- LDAP is oftentimes a lightweight database well suited for simple, lightweight, data storage that is easily accessible by many Internet-based applications; this at least partly implies that a large database system is not needed, though one could probably be used; it should be noted, however, that there have been some very large LDAP databases implemented
- data inquiries are relatively simple, especially when compared to some types of commercial database systems
- methods developed to access an LDAP database are not limited to one particular language or compiler
- as more Internet-based applications become LDAP compatible, they will increase the number of applications that can be easily integrated with other Internet-based applications using common data in an LDAP server
- LDAP provides a fairly nice data structure that lends itself well to organizing certain types of data, most of which relates to people, processes, organizations, and services
- LDAP databases are easily extendable in their data types and data structure
- small to medium sized LDAP servers can be implemented on small to medium sized computers quite easily, which may seem trivial until compared with the size of machines dedicated for large database systems
- LDAP can be implemented on commercial computer systems or free UNIX-like systems, which may lead to sizable cost benefits

4 Project Overview

The following is a brief overview of the three major and two minor uses of LDAP already mentioned. The later additions will be presented first since they are short and simple.

- a simple script to query LDAP database for LAN Admin information

  Having the requisite portions of the LDAP suite installed locally, a short line-mode Perl script was developed to perform an LDAP search for the Administrator of a LAN, based on one of several types of identifiers. The purpose is to provide a simple, easy, and quick method to look up the LAN Administrator for a given LAN, providing their name, phone number, and electronic mail address.

- a simple LDAP search via a short JavaScript-based Web page

  The intent here is to use an LDAP-enabled Web browser to initiate an LDAP inquiry without requiring any CGI scripts or other services from a Web server. The Web browser itself acts as the sole LDAP client. A simple HTML form is used to obtain the LDAP search item, and a simple JavaScript function then converts that into an LDAP URL[16]. When this is inserted into the browser's location field, the LDAP search is performed.

- LDAP implementation with Sendmail source and LDAP enhancements to Sendmail process

  Using LDAP required two enhancements to Sendmail: using internal hooks in Sendmail for LDAP; and an LDAP-based mail delivery program, *mail500*[12, 15], external to Sendmail. With these extensions, simple virtual mail users with different address formats are easily created and serviced. Once Sendmail is installed with the required LDAP libraries, a small set of code is added to the Sendmail configuration file to enable both enhancements. In one case, the mail was handed off to the *mail500* LDAP tool, which performed the final delivery. In the other case, Sendmail itself connected to the LDAP server to find the requisite information to complete the mail delivery that it would perform.

- enhancement to INN (InterNetwork News) via a Perl script

  The InterNetwork News product provides a simple security scheme to control access to newsgroups[10]. More finely detailed access methods are not directly available with the basic INN program. Starting with version 2 of INN, a Perl "hook"[8, 9] is provided to invoke user defined code. Using this Perl script and data stored in an LDAP database, a finer grained access method is now possible. It also provides the ability to define a multi-tiered access method. In this report, this method is used to control authorized postings to a local newsgroup that did not use the network news "moderated" format and controls.

- implementing a mainframe-to-PC File Relay using Samba, Expect, and LDAP

  This facility used a UNIX server running Samba, the OpenSource UNIX NetBIOS file system service, to provide a type of automated file relay between a Windows PC and a mainframe. In reality, the PC user had two network drives, an "input share" and an "output share," that appeared to connect them directly to the mainframe for simple output retrieval and job submission. LDAP stored user specific information to be used by the FTP process to the mainframe. Expect was used to automate the process of FTPing a user "job" to the mainframe for execution from the "input share," and keeping a log of the FTP process. Mainframe output was FTPed to the "output share" by some additional job control language statements that executed an outbound FTP session from the mainframe to the Samba server.

4.1 Environments Used

There were three network environments used for different stages of developing these facilities. The statistics listed elsewhere will be affected by the capabilities of the different networks and hosts listed below. Note that system loads are not reported in any of the statistics.

- development and test environment (#1)

  LAN: 10Mb/s Ethernet

  LDAP/Sendmail server: 50MHz Motorola 68060 CPU, 32MB RAM, 10000rpm SCSI HD, Amiga A2000 workstation

  INN server: 40MHz Motorola 68040 CPU, 32MB RAM, 7200rpm SCSI HD, Amiga A2000 workstation

- test environment (#2)

  LAN: 10Mb/s Ethernet

  INN/Sendmail server: 400MHz Intel Pentium II CPU, 128MB RAM, 7200rpm IDE HD. Dell OptiPlex GX1 workstation

  LDAP server #1: 400MHz Intel Pentium II CPU, 128MB RAM, 7200rpm IDE HD, Dell OptiPlex GX1 workstation

  LDAP server #2: 70MHz Fujitsu microSPARC II CPU, 64MB RAM, 7200rpm SCSI HD, Sun SPARCstation 5 Model 70 workstation

- production environment (#3)

  User LAN: 10Mb/s Ethernet

  Server LAN: 100Mb/s FDDI

  LDAP server: 167MHz Sun microSPARC II CPU, 256MB RAM, 7200rpm SCSI HD, Sun Ultra Enterprise 2 server

  INN server: IBM RS/6000 CPU, 192MB RAM, 7200rpm SCSI HDs, IBM PowerPC Model 250 server

5 Simple Perl Script To Perform LDAP Query For LAN Admin

The LDAPSEARCH tool is a line-mode access tool that sends an LDAP query to an LDAP server, and displays all of the data returned in ⟨key⟩⟨value⟩ pairs. The actual LDAP search is a one-line command. The Perl script enables the user to identify something about the desired LAN Administrator such as IP subdomain name, or AppleTalk network number range. The script then calls the LDAPSEARCH tool to talk to the LDAP server and get the desired data, which was limited by a set of search command, key selectors. The results are displayed as a simple line-mode table, as shown in the example in Figure 1.

While the LDAPSEARCH tool performs the actual LDAP lookup, its command format can be very long, tedious to enter, and not pleasing to the eye to behold. Also, LDAP results are usually one or more lines of ⟨key⟩⟨value⟩ pairs that usually have a raw appearance. The Perl script performs all of the work to make the search process simple and effective, and provides a better display.

Figure 1 is a sample result of the *QLANADMIN* Perl script used to provide this service, searching for the LAN Administrator of a specific IP subnet. When no parameters are given, the script will display information on how to use the script, listing the various parameters accepted. Figure 2 is a sample of the the LDAPSEARCH command used and the data returned from the LDAP server.

The LDAPSEARCH command is given several "selection attributes" that it passes to the LDAP server to restrict the number of attributes that will be returned by the server. In this script, the four selection attributes specified were: *lanadmin postofficebox telephonenumber dnsadmin.*

```
qlanadmin 216.000

====================================
Subdomain        : grdsch
Network Protocol : IP
LAN Administrator: FirstName LastName
Department       : Department Name
E-mail Address   : userid@mail.host
Telephone Number : (xxx) xxx-xxxx
DNS Administrator: FirstName LastName
======================================
```

Figure 1: QLANADMIN

```
ldapsearch -h ldap.hostname -b o=ournet,
  c=us "subnet=216.000" lanadmin
  postofficebox telephonenumber dnsadmin

dc=grdsch,dc=ournet,dc=edu,o=ournet,c=US
lanadmin=cn=LAN Admin name,
  group=employee,ou=orgName,o=ournet,c=US
postofficebox=userid@mail.host
telephonenumber=(xxx) xxx-xxxx
dnsadmin=cn=DNS Admin name,group=employee,
  ou=orgName,o=ournet,c=US
```

Figure 2: Raw LDAP search command and response

The distinction between IP subdomains and AppleTalk zone objects is made in the DN entry. Other attributes within each entry could have been used that are not a part of the DN. There are trade-offs to both approaches, but they won't be covered here.

As it was implemented, IP subdomains were represented by a DN which contained a Domain Component (DC) attribute for each component of the subdomain name. In the example above, the IP subdomain `grdsch.ournet.edu` has a DN that begins with `dc=grdsch,dc=ournet,dc=edu`. A representative AppleTalk zone is similarily identified, but only uses two DC attributes: one for the actual AppleTalk zone name, and one for a pseudo domain of `AppleTalk`.

Late note: multiple LDAPSEARCHs may, or may not, be better acccomodated with `perldap`[20].

5.1 Performance Observations

Two versions of the *QLANADMIN* script were created: one using Perl and one using REXX[19]. The test runs obtained and displayed the same LDAP data, and were run on the LDAP server in test environment #2 but used the production environment LDAP server for data lookups. "u" and "s" refer to user and system CPU usages, in seconds.

Test Performed	*Perl Script*	*REXX Script*
size	3,178 bytes	2,879 bytes
Perl Validate	0.012u,0.006s	-na-
display results	0.013u,0.014s	0.009u,0.015s

Table 1: Sample QLANADMIN Performances
Test #2 and Production Environment

6 Simple LDAP Query Via Short JavaScript Based Web Page

A simple Web page instructs the user to input part of an IP address or hostname, which is entered into an HTML text input field. Upon clicking on the Search button, the embedded JavaScript function creates an LDAP URL[16], in simplified and sample forms:

```
ldap://<LDAP server hostname>/
   <LDAP search base><LDAP search string>

ldap://<LDAP server>/o=siuc,c=us??dc=<hostname>
```

The JavaScript function then changes the browser's current document URL, or "location field," to the composed LDAP URL which causes the built-in LDAP client to perform the LDAP search. Note that this requires the equivalent of Netscape V4 or higher.

The browser displays the LDAP results in a raw format as seen in Figure 3. The entire record returned by the LDAP server is displayed except for those fields that are restricted by LDAP access controls.

```
Object Class   top
               domain
               localadmin
dc             ournet
Notes          Internet network domain
                      for our.network
associatedname our.network
City           location
Organization   department name
postofficebox  mailbox@mail.host
subdomain      sub.domain.name
subnet         999.999
dnsadmin       LDAP DN (which includes
                       user name)
lanadmin       LDAP DN (which includes
                       user name)
creatorsname   LDAP Manager DN
modifiersname  LDAP Manager DN
createtimestamp19990519180831Z
modifytimestamp19990519180831Z
```

Figure 3: Sample Raw Web-based Data Display

Since there were only two possible choices for the LDAP search string, or target, a simple `if ...else` statement is all that is necessary to create the dynamic portion of the LDAP URL. In the end, a fixed URL prefix was concatenated with the determined URL search target suffix obtained from the input field, for example ("+" is JavaScript string concatenation):

```
URLsuffix = "dc=" + form.SearchData.value;

document.location = sSearchURL + URLsuffix;
```

which is then inserted into the browser's "location" field, causing the built-in LDAP client to perform the LDAP lookup.

The returned data is neither formatted nor limited, as it was in the case of the *QLANADMIN* Perl script, other than being limited by access controls on the LDAP server (see Section 10 for more information). This may allow more data to be displayed than needed or anticipated and may not be as visually appealing as desired.

Other LDAP search tools such as Web500GW, Wax/Max500, or Netscape Directory Server V3/V4 gateway (similar to Web500GW) might format the returned LDAP data differently. In some cases, they don't – it has the same appearance as the raw format. They also may provide special functions for some of the attributes, such as providing an automated e-mail function upon clicking on the e-mail address. Any URLs in the returned LDAP data may also be turned into active URLs (i.e., clicking on them goes to the specified Web page).

This example Web page facility demonstrates a uniqueness about LDAP-enabled Web browsers in that it does not depend nor rely upon a Web server for any assistance. There is no CGI script that gets executed to receive the search data, perform the LDAP search, create a new Web page for the data returned, and send the new Web page back to the client for display. The LDAP-enabled Web browser is able to display the LDAP search result all on its own. For a very simple search tool that is easy to develop and use, the raw data format may not be pretty, but may suffice – at least for testing.

Kartik[18] has a nice example of an extensive JavaScript/HTML tool for administrative LDAP searches and database updates, also without the need for a Web server.

7 Sendmail Enhancements Using LDAP

When Sendmail Version 8[3, 4] is compiled with UMich LDAP-3.3 or OpenLDAP (or possibly Netscape Directory Server) libraries, it has direct access to an LDAP database. Sendmail[4, 5] also comes with *sendmail.cf* configuration file examples for using this LDAP access. While Sendmail configuration programming is considered a black art in some cases, the enhancements discussed later used only a small number of additions or changes to the *sendmail.cf* file to provide the LDAP services previously mentioned.

In addition to Sendmail source code that provides access to LDAP databases, both the UMich LDAP-3.3 and OpenLDAP distributions include useful and important LDAP client contributions, one of which is related to electronic mail and Sendmail: *mail500*. *Mail500*[12] is used to provide external access to an LDAP database. Sendmail passes pending messages off to *mail500* which does all of the work of extracting e-mail addresses from the LDAP server. *Mail500* then passes the messages back to Sendmail for final delivery with the LDAP extracted e-mail addresses in place.

Sendmail can make use of many user-defined databases, which usually are either regular "flat files," or "hash mapped" database files. Using LDAP is not necessary to provide for virtual e-mail users, but makes the process more flexible as well as eliminating constant changes to Sendmail configurations and/or databases.

To use the LDAP function from within Sendmail, an LDAP "database map" and "relay host macro" definition is added to the *sendmail.cf* configuration file, along with a few additional lines in Sendmail's rule set #5. Using the external LDAP function via *mail500* required a one-line addition to Sendmail's rule set #98 and a two-line "mailer" definition for the *mail500* external "mailer" program. The *sendmail.cf* "rule sets" are the means by which Sendmail determines what to do with any given piece of mail.

Each set of additional lines defined a specific character string which when matched to a supplied e-mail address, would invoke the respective LDAP-enabled service. Many trigger combinations are possible, but this implementation limited itself to just the two sets detailed later. The intent is to provide a simple yet effective means of selecting e-mail addresses from a fixed set of virtual e-mail users.

Virtual users, in the context of this facility, are considered to be users whose e-mail addresses map to the local Sendmail server but whose userids (left-hand portion of each of the e-mail addresses) are not found in the Sendmail server's password file. With the modifications mentioned in place, if Sendmail fails to resolve a local userid via its password file, it then calls upon the defined LDAP server to retrieve an e-mail address from the LDAP database. In this manner, many users who have an LDAP entry based on a userid or other attribute which matches the left-hand portion of an e-mail address, and who have an LDAP e-mail attribute defined, can be considered as local to the Sendmail server.

With the aid of LDAP, the number of the e-mail users, their identities, and their final e-mail addresses to be used for mail delivery need not be known in advance by Sendmail, thus obviating work required to create "static" user definitions for special purposes. No additional Sendmail configuration is necessary other than what has already been mentioned. Nor is any additional Sendmail configuration necessary when virtual e-mail users change, so long as the required LDAP attribute names do not change.

In Sendmail's rule set #5, one rule compares the given e-mail address with a specific format that separates the left-hand side of the address from the rest. This rule set is used to perform some specific tests on e-mail addresses that are determined to be local to the Sendmail server. One part of it attempts to locate the userid extracted from the e-mail address in the mail server's password database. This occurs in the section labelled as, "send unrecognized local users to a relay host." If the userid is not found, it will later be passed to another Sendmail rule which will attempt to send the pending mail to another mail server, defined as a mail relay.

This implementation inserts a Sendmail rule after the above section to then attempt to look up the userid in the defined LDAP database. Here, the modified Sendmail program itself is the LDAP client. If the database search returns an e-mail address, Sendmail then uses it in place of the original address and eventually attempts to deliver the pending mail to the new e-mail address. This provides for a virtual user who is not directly associated with or defined on the host running the Sendmail server but whose e-mail address conforms to the normal format for that particular server.

In rule set #98, a new rule was added to select processing by the *mail500* program. This rule set is part of the Sendmail logic used to determine how the actual mail delivery will be completed, and by which program. As a part of this processing, an external program, such as *mail500*, may be selected to complete the mail delivery.

In this implementation, the rule added to rule set #98 selects mail based on the following pattern, which matches one of the LDAP *cn* entries as shown in Figure 4:

```
<firstname>.<lastname>@<Sendmail server E-
mail domain>
```

Now, mail with the indicated address format will be directed to *mail500* for a simple LDAP CN attribute lookup based on the `<firstname>.<lastname>` portion. If a corresponding *mail* attribute is found, *mail500* will then formulate a new address and pass the mail back to Sendmail for final delivery. This is expected to return a single e-mail address for one person.

Another rule, added later on to rule set #98, selects mail based on the next pattern to extract multiple e-mail addresses associated with a mailing list.

```
<mailing list name>.mlist@<Sendmail server E-
mail domain>
```

The rule strips off just the mailing list name, and passes it to *mail500*, along with the contents of the message. *Mail500* then searches for a CN attribute belonging to an LDAP *rfc822MailGroup* object that matches the mailing list name. If found, it then scans the LDAP object for all *mail* and *member* attributes, using the corresponding e-mail addresses to form a single mailing list. When all of the addresses are found, *mail500* passes the message back to Sendmail with the list of new addresses, and Sendmail finishes the mail delivery, assuming no other matching process occurs with the new e-mail address list.

Part of an LDAP entry that works with these methods might look something like the following:

```
dn: cn=Jim Dutton, ou=<deparment>,
    ou=People,o=<our domain>, c=US
objectclass: ...
cn: Jim Dutton
cn: jim.dutton
cn: Jim_Dutton
cn: jimd
sn: Dutton
uid: <userid>
mail: <userid>@<hostname>
```

Figure 4: Sample LDAP Entry For Sendmail Use

For the LDAP database lookups this facility chose to use the LDAP Common Name (CN) attribute to search for an entry that would have a *mail* attribute defined. The left-hand side of the e-mail address is extracted via the applicable Sendmail rules. This is then used as the key value for the LDAP CN search. The LDAP search will then return the value of the *mail* attribute if the corresponding CN attribute is located and it has a *mail* attribute defined. Another LDAP attribute could have been used to provide the LDAP search key, but it must be related to an LDAP entry which will return a valid e-mail address.

With LDAP, some attributes can have multiple instances whereas others normally cannot. The Common Name attribute is one that is normally allowed to have multiple instances, or occurances, of attribute values. Since this does not require any special LDAP configuration to use, it is easy to make use of in this facility, and in other situations. In the above sample, multiple CN attributes are defined including one with the specific $\langle$*firstname*$\rangle$.$\langle$*lastname*$\rangle$ format. This provides for multiple versions of a "name" attribute which can be used to locate a specific LDAP object, which doesn't necessarily have to be a person.

While an LDAP server also provides for internal substring matching of search keys, thus providing for entry matches based upon part of an attribute value but which can be quite costly to perform, using multiple CNs provides for a set of easily matched and defined qualifiers that may be considered alternative spellings of a primary Common Name. The use of multiple CNs, and some other attributes, increases the probability of an LDAP search hit including the possibility of providing for misspellings of the primary attribute. This becomes very apparent when a user's name as used for the LDAP DN is not the same as the user's name as used in practice. Without additional search qualifiers (i.e., multiple attributes), searches for the specified DN may indeed become difficult and frustrating. This is one of the important benefits of the LDAP data structure and usability.

7.1 Performance Observations

Test Performed	*LDAP Lookup*	*Mail Delivery*
1-user LDAP	$\leq$ 2 secs	2 secs
1-user direct	-na-	$\leq$ 1 sec
5-user list	$<$ 3 secs	5 secs

Table 2: Sample Sendmail Performances
Test #1 Environment

The 1-user note size 902 bytes, including all SMTP header records and one text line. The second test was sent directly to the same mailbox acquired in the first test from LDAP. The 5-user mailing list was directed to 2 users with *mail* attributes and 3 users with *member* attributes in the mailing list LDAP object. The *member* attributes where subsequently looked up in the LDAP database for their related *mail* attributes. The mailing list note delivered was 868 bytes in size, including all SMTP header records and one text line. The mail log entry for the mailing list contained extensive debug statements that would not normally be present.

8 INN Enhancement Using LDAP

InterNetwork News (INN) is server software used to provide Network News, or Usenet, services. The host which provides this service is usually referred to as the news server. In many cases, an Internet Service Provider (ISP) will automatically provide Network News service to new customers, giving them Read and Post access to the all of the newsgroups, sometimes also known as "discussion groups," that the ISP provides via their news server.

For the most part, the general newsgroup article reading and posting access privileges are sufficient, and no additional access mechanism is necessary. Combinations of the fields in the INN access security file, *nnrp.access* (see Figure 5), provide the basic capability to allow or disallow access to a newsgroup or the news server itself, and allow or disallow reading and posting of a news article. In some cases, however, this simple security scheme is not sufficient to enable more complex access criteria, or to provide for other qualifiers for access control.

Version 2 of INN introduced a Perl script called by *nnrpd*, the server program that handles users newsgroup accesses. It is referred to as a "hook" since the *nnrpd* program automatically executes the specific Perl program named *filter_nnrpd.pl*, if it exists. The function of this hook is to allow the local news administrator the opportunity to code whatever conditions are necessary for establishing a user's right to post an article, thus providing an extended access control function. With the basic INN user access control, article posting is either "yes" or "no" from the time the user connects to and is accepted by the news server. With the *filter_nnrpd.pl* Perl hook, the news server can now say "maybe" to an attempt to post an article, and then make a more informed decision based upon the final outcome.

INN uses a file named *nnrp.access*, whose syntax is described in [10] and illustrated in Figure 5, to control access to newsgroups. Newsgroups may be defined as "moderated" or "not moderated." Most generally available newsgroups are of the "not moderated" variety. To be a "moderated" newsgroup means that article submissions are sent to one or more persons via electronic mail, rather than being posted to the specified newsgroup. These "moderators" then review the article-to-be-posted, and if they accept it, complete the posting process (which will not be discussed here).

The goal was to provide a local newsgroup, one that is not distributed across the world, which had three special needs. The first was to allow any user with access to the newsgroup the right to post a new article (i.e., one that has no previous references). This fits into the basic INN access method whereby "Read" would be included in the *permissions* field of *nnrp.access*. However, the basic INN "Post" permission does not provide for any distinctions about what, who, or when an article may be posted. The permissions of "Read Post" allow anyone who can access the newsgroup to read any existing article, and post any desired article at will to that newsgroup, without any limitations. For a new article, the basic INN access permissions are just fine.

```
<F#1>:<F#2>:<F#3>:<F#4>:<F#5>

<F#1> = IP hostname or address of user’s
        IP host; may be empty

<F#2> = newsgroup access permissions
        R(ead) P(ost); may be empty

<F#3> = username user will be known as when
        using this service; may be empty

<F#4> = password user will use when using
        this service; may be empty

<F#5> = list of newsgroups accessible for
        this particular user/host
```

Figure 5: INN *nnrp.access* file

The second special need was to implement selective posting of an article that referred to a previous article, usually referred to as a “followup.” The *filter_nnrpd.pl* hook is used to make the final decision of whether to accept the article or not. Specific project instructions were given that identified only a select handful of people who were given the referenced article (followup) posting permission. They and they alone would be able to reply to an existing article with a followup article, and were to be defined as “authorized reply posters.” All other users would be denied this privilege.

The third special need was to allow authorized reply posters the ability to post followup articles without having to use the “moderated” control scheme, which is the basic INN method to control article posting. This need was taken care of as a side affect of the solution for special need #2.

An LDAP database is used to store the list of authorized reply posters, including information used for security purposes, and other information about the special newsgroup. When the news server receives an article from a user to be posted, it invokes the *filter_nnrpd.pl* hook. If the pending article is destined for the special local newsgroup, the Perl script uses the LDAPSEARCH tool to obtain the authorized reply poster list and their associated security data. If the pending article is a new article, it is automatically accepted. If it is a reply, or followup, article to the special local newsgroup, the article author is verified against the authorized reply posters list and their associated security data. If there is a match, the article is accepted and subsequently posted.

The choice of using an LDAP database is mostly to give the owner of the special local newsgroup the ability to make changes to the authorized reply posters list without requiring any changes to INN (including the Perl hook). It also is meant to take advantage of existing LDAP entries for the individuals responsible for the local newsgroup, and entries for locally provided services (i.e., Network News). Using LDAP also means that the news administrator does not need to perform any ongoing maintenance to INN or *filter_nnrpd.pl* to keep the list up to date. It also provided ancillary information about the authorized reply posters that would not have been possible with just a hard-coded userid in *nnrp.access*.

The *filter_nnrpd.pl* script, as implemented for this facility, is set up with a Perl array to specify more than one LDAP server. This assures access to an LDAP server in the event that one or more of the servers defined is unavailable for service. Required LDAPSEARCH command parameters and LDAP search parameters are stored in Perl variables to make LDAPSEARCH command execution simpler to code. A local log writing facility, *logger*, is called upon to enter lines in the *news.notice* log file to show important operations of the script.

To obtain the list of authorized reply posters, the LDAP search base is set to the LDAP Common Name of the object representing the special newsgroup. The list of attributes to return is assigned to the *attr* variable as,

```
        $attr = "certifiedAuthor";
```

and the LDAP search is performed, using the following Perl statement,

```
@result = ‘$ldapcmd $ldaphost[$i] $parm
   $scope "$base" "$filter" $attr‘;
```

where predefined Perl variables are used for the command and its parameters. The LDAP search results are put into the `result` array, which is then searched for the desired information. Access to the data is controlled by the LDAP ACL as shown in the first ACL example in Section 10 (Figure 8) where the LDAP server receives a network connection from the news server and matches its IP address against the *by addr* portion of the ACL. Using information obtained from the pending news article and the LDAP server, the script determines whether the article will be accepted for posting.

Using the Web500GW[11] Web interface, the special newsgroup owner can readily see and change the LDAP data relevant to their newsgroup. They are then able to manage the authorization list without any further work by the news administrator. With the Perl hook and coding added to access the LDAP database, simple or complex decisions can be made with regards to specific article postings. This then extends the levels of security normally available to INN without requiring local modifications of the INN programs themselves.

8.1 Performance Observations

While not exhaustive, the following simple data may help to show some of the performance differences between using an INND server without any LDAP enhancements, and an INND server using the enhanced *filter_nnrpd.pl* filter with multiple LDAP searches. The "Prod. Script" uses LDAP and extra coding whereas the "Orig. Script" that comes with INN basically does nothing.

Test Performed	*Prod. Script*	*Orig. Script*
size	14,091 bytes	1,207 bytes
Perl Validate	0.548u,0.233s	0.127u,0.180s
One line Post-s	0.975u,0.747s	0.404u,0.363s
One line Post-u	$\leq$ 6 secs	$<$ 0.75 sec
NNRPD write/log	$<$ 4 secs	-na-
NNRPD reject/log	$\leq$ 6 secs	-na-
LDAP search/log	$\leq$ 3 secs	-na-

Table 3: Sample INN/LDAP Performance
Test Environment (#1)

Test Performed	*Prod. Script*	*Orig. Script*
size	14,100 bytes	1,207 bytes
Perl Validate	0.180u,0.060s	0.010u,0.050s
One line Post-s	0.400u,0.380s	0.180u,0.280s
One line Post-u	$\leq$ 2.25 secs	$<$ 0.5 sec
NNRPD write/log	$\leq$ 2 secs	-na-
NNRPD reject/log	$\leq$ 2 secs	-na-
LDAP search/log	$<$ 2 secs	-na-

Table 4: Sample INN/LDAP Performance
Production Environment

The "u" and "s" symbols in the times represent user and system CPU usages in seconds, respectively. Those entries which are not applicable since the script does not perform or have any impact on the indicated function are identified with "-na-."

The Perl validation is the simple *perl -c* test which validates the Perl code (i.e., performs the Perl "compile" operation) which is affected by the size of the code being tested and the complexity of the code. The original script contains only a few active lines – the remaining being either blank or commented out.

The "One line Post" times come from a new article posted to the special test newsgroup that contained one line of text, which varied slightly in size, plus the normal article header records. The "-s" times are news server times for just the *nnrpd* program itself. The "-u" times represent the real-time measures of time from when the user initiates the posting action (i.e., presses the Post button, or equivalent), and the article is posted (as reported by the user's news client).

All "/log" entries represent the elapsed time observed for the particular function performed, reported as integer values. Logging is accomplished with the system *logger* command, where each log line is a seperate call to *logger.* Time values indicated as "less than" are strictly less than the value indicated. Values indicated as "less than or equal to" are predominantly less the the value indicated – there were a small number of observations that did equal the specified value.

"NNRPD write/log" times represent the total elapsed time for *filter_nnrpd.pl* to write its entries to the log file after the user posts the article (i.e., presses the Post button, or equivalent), and *filter_nnrpd.pl* accepts it.

"LDAP search/log" times are taken from an LDAP server log which show the total elapsed time for the LDAP server to accept a connection from the news server and process its LDAP search requests for the special newsgroup entry, return the data to the news server, and close the network connection.

"NNRPD reject/log" times are from the news server log which show the total elapsed time for the *filter_nnrpd.pl* "hook" to make its posting failure determination and write out its log lines.

The differences in log file writing times are affected by the throughput of the associated network and the speed of the associated servers, which is to be expected. The different environments, slower test, faster production, also represent a difference in the relative "size" of the servers (i.e., small versus medium). The amount of *filter_nnrpd.pl* logging appears to be the largest delay factor in this process, all other things being equal. It should be noted that *filter_nnrpd.pl* can operate without generating any log entries, but for administrative purposes, ten or so lines were normally logged.

Overall, while increasing news server system time for log writing, Perl "hook" processing, and external service (LDAP) processing, the amount of additional time for the LDAP-enhanced INN services is reasonable. Turn off all *filter_nnrpd.pl* logging and the additional time can be further reduced, making it neglible.

9 Implementing a PC-to-other host File Relay

The main goal of this facility is to provide an easy way for a PC user to submit jobs to a remote host and obtain output from a remote host, without the PC user having to connect or logon to that remote host. All of the file transfer exchanges appear to be local to the PC user. This eliminated the need for the PC user to stop what they are doing, initiate an FTP process to the remote host, and then issue one or more commands to send or receive a file. Occasionally, a login or telnet session would also be required to obtain the desired output. All they need to do now is to copy a file from or to two network drives (remote Samba server "shares"), and everything else is done automatically.

While this particular facility was developed for use with an IBM mainframe and NetBIOS/Windows PCs, it can be equally used for other hosts where a remote job could be executed and file transference is accomplished by FTP (or some other suitable function). Implementing NetaTalk[21], the UNIX-based AppleTalk file system, this file relay is also available to Macintosh hosts. Again, while the remote host details will be specific to a mainframe in this project report, the process is not limited to such a host. The same is true for the user side. In the remainder of this section, the terms "PC user" and "PC host" are interchangeable with other user-based scenarios using different types of computers.

The key mechanisms used for this file relay are FTP and user workstation file copy. The remote host is expected to be an IP host which provides an FTP server to allow files to be sent to it. In the case of the IBM mainframe, a special FTP command (SITE) was used to direct the incoming file to the "job queue" of the remote host. To obtain remote host output, the user would add several FTP commands to their "job" which executed on the remote host. This would send the output data to the Samba server. To obtain a local copy of the output data, the user simply copies the file from their Samba server "output share" to a local folder.

It should be noted that the LDAP service was chosen to store the user's remote host FTP/login password, but the same data can be supplied in the user's job file. In general, however, it is simpler to ask an LDAP server for a data entry than to parse a job file, looking for the same data even if a fixed scheme is applied and expected in locating said data. The LDAP mechanism also allows more data to be stored for the given user, such as a "notify" field intended to be used to send a notification to the user in certain situations. Again, this same information could be incorporated into the job file to be submitted, which then must be parsed and properly extracted. The LDAP mechanism helps to reduce the multiple possibilities of user data entry errors, though it certainly does not eliminate all possible user errors.

Figure 6 and Figure 7 illustrate the before and after affects of providing this service.

In Figure 6, the user must manually go to the remote host to submit a job or retrieve job output. This may or may not require a telnet session to find the job output information. The user must be familiar with the FTP process including all special commands such as the SITE command required in our situation. They must also be familiar with finding their output data files and how to move them to where they want them, most likely using an FTP session. For some users this process may be counterintuitive and time consuming.

In Figure 7 the user has established two network drives available to them from the Samba server, which appear as folders on their local system. They can then copy files in to and out of these network drives as easily as they would with any other local folders. This provides a well understood mechanism for the user once they understand which files go into or can be found in which remote folder. They also are free to use whatever local tools they desire to do the file transference required.

The job to be run is copied into the remote submit folder on the Samba server. An automatic process added to the Samba server, using Expect, then sends the job file to the remote host for execution via FTP. Two log files are maintained: one in the user's submit folder which shows the submission process result, and one in the server's log directory which shows a count of all jobs submitted per user.

A special "job step" can be used to transfer output data to the user's remote output folder on the Samba server. This can be added to a job file placed in the submit folder, or can be a job that the user has permanently stored on the remote job execution host. Upon execution of this special job step, the remote host deposits the specified output file into the user's output folder. The user can then copy the output file to their PC.

Using this process, the user never has to logon or connect to the remote host.

10 Brief Comments On LDAP ACLs (Access Control Lists)

Netscape Directory Server, UMich LDAP-3.3, and OpenLDAP all use a scheme to control access to parts of the LDAP database by specifying one or more internal database attributes, or external real-world attributes. Netscape has renamed ACL to ACI, but the mechanism is basically the same. The biggest difference between Netscape's ACI and LDAP ACL is where the actual control list resides. With Netscape's Directory Server, the ACIs are entries within the LDAP database, usually at the root of the database tree to be controlled.

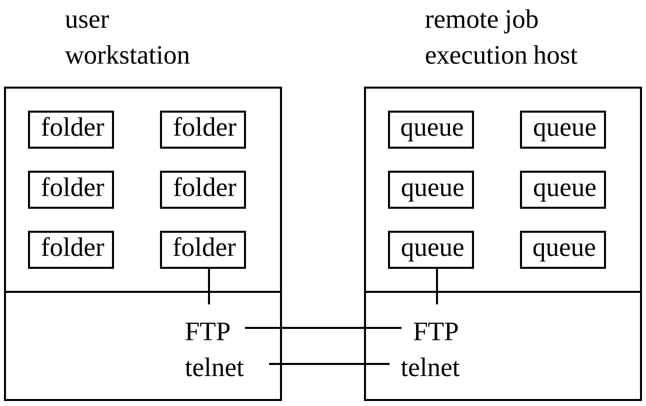

Figure 6: User to remote host via FTP and telnet
User does all of the work

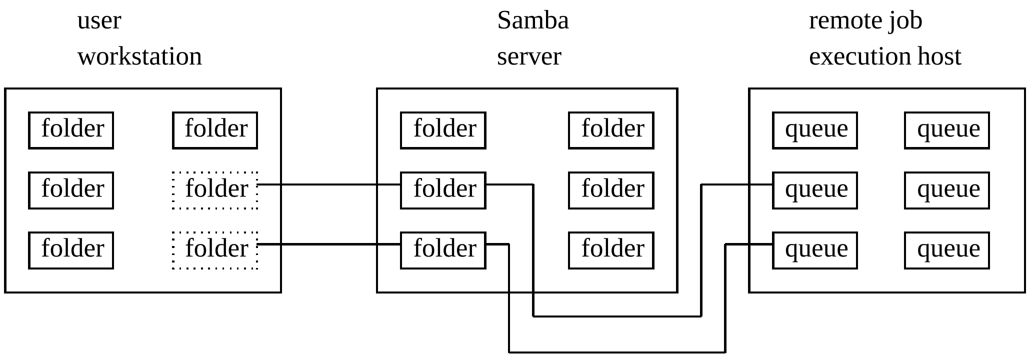

Figure 7: User to Samba server
Samba server does all of the work

With UMich/OpenLDAP, the ACLs are usually stored in an external file which is included in the LDAP server startup file.

The basic LDAP ACL definition[7, Section 5.3] is shown below. Netscape's ACI has additional attributes.

```
<access directive> ::= access to <what>
               [by <who> <access>] +

<what> ::= * | [dn=<regex>]
               [filter=<ldapfilter>]
               [attrs=<attrlist>]

<who>  ::= * | self | dn=<regex> |
               addr=<regex> |
               domain=<regex> |
               dnattr=<dn attribute>

<access> ::= [self]none   | [self]compare |
             [self]search | [self]read |
             [self]write
```

Samples of ACLs used in the facilities of this report, with some specifics replaced by generic qualifiers, are shown in Figure 8.

```
access to filter="cn=<newsgroup name>"
     attr=certifiedHost,certifiedAuthor
  by self write
  by dnattr=owner write
  by dn="cn=Manager,ou=People,
     o=<our domain>,c=US" write
  by dn="cn=<News Server IP hostname>,
     ou=Network Hosts,o=<our domain>,
     c=US" read
  by addr=<News Server IP address> read
  by * none

access to attr=MVSpasswd
  by self write
  by dn="cn=Manager,ou=People,
     o=<our domain>,c=US" write
  by addr=<Samba server IP address> read
  by * none
```

Figure 8: Sample Project ACLs

The first ACL restricts all access to the *certifiedHost* and *certifiedAuthor* LDAP attributes defined in the LDAP entry for the special newsgroup whose DN begins with the Common Name value of `<newsgroup name>`. Write, and read, access are granted to the person identified by the *owner* attribute, which is a "pointer" to another LDAP entry (DN). The person associated with the (directory server) Manager DN is given the same access permission.

Access for these privileges requires LDAP password authentication, where the password is stored in the LDAP entry of the authenticating user. The news server using the LDAP database is allowed to read the restricted attributes by either authenticating as the news server LDAP entry DN, or by its IP address matching the value in the *by addr* specification. All other access attempts for the two specified attributes is denied.

The second ACL in Figure 8 restricts access to the *MVSpasswd* attribute defined in any LDAP entry. The previous ACL was limited to a single LDAP entry. Here the user (*self*) identified by their entry's DN and the LDAP administrator both have write and read access. The Samba server which uses the *MVSpasswd* data is allowed only read access, and its IP address must match that as given in the *by addr* specification. All other access attempts are denied.

After the owner of any restricted LDAP entry becomes familiar and comfortable with the process, the LDAP administrator may or may not be removed from the specific ACL. It is usually included initially to get the new service operating, and to provide a fall back in case the owner has problems updating their restricted data.

11 Conclusions and Comments

While many uses of LDAP data and servers today relate solely to electronic mail addresses, one of the inherent benefits of using LDAP is the ability to easily relate many other informational items with any given LDAP entry. This can greatly expand the knowledge base associated with a person, group, service, network entity, organization, or other type of object. This is not to say, however, that LDAP is a ready replacement for all uses or forms of other databases.

One major difference between Sendmail's LDAP client capability and the *mail500* LDAP client program is that Sendmail only accepts one returned e-mail address whereas *mail500* accepts all returned e-mail addresses. This is not a weakness in Sendmail's LDAP client, but rather a difference in philosophy which can be put to good use. If a note to a mailing list could generate many e-mail addresses, then allowing an external program such as *mail500* to do all of the work in assembling those addresses from an LDAP database relieves Sendmail of work that it doesn't really need to do, freeing it to do the work it is more expected to do – deliver mail. On the other hand, a single user LDAP lookup fits in quite well with Sendmail since it can, and sometimes does, the same kind of process in other user databases created for Sendmail specifically. It should be noted that the specific uses of LDAP in this report are not the only way that the desired affects could have been achieved.

In general, data stored in an LDAP server that is meant to be user consumable is in clear text – not encrypted. Internal server information may or may not be encrypted, but the user does not see and usually cannot access such data. Also, most general LDAP connections today are insecure, not only for connection or update authentication, but also for data transmission. Additionally, there may be possibilities of IP hostname or address spoofing. These can be serious concerns for some sites or individuals.

In the case of the file relay facility, some possible methods to increase security are:

- encrypting user's remote host login password in the LDAP database,
- encrypting data transferred between the LDAP server and the submission client,
- using a Kerberos-enabled FTP client in conjunction with a Kerberos-enabled FTP server.

However, in the end, the file relay service is dependent on trust and accountability no matter what security mechanisms are used since its job submission client must deal with a remote host userid and password, or equivalent, to perform its function.

The facilities in this report were not implemented with the intent of providing secure applications. They are meant to show the feasibility of certain LDAP-enhanced applications, and to provide a test bed for exploring such LDAP add-ons.

Version 3 of the LDAP protocol specifies a secure BIND capbility through the use of Simple Authentication and Security Layer (SASL), RFC 2222. Several INTERNET-DRAFT documents propose the use of Transport Layer Security, RFC 2246, as a means of protecting data transmitted between an LDAP client and server. It may be possible to create "secure tunnels" or "wrappers" using OpenSSH, OpenSSL, SSLeay, and/or `stunnel`[22] for LDAP clients or servers without security capabilities.

Perhaps in the future, LDAP clients and servers will have the requisite hooks built-in or coding added to provide security at all levels of interaction. In the meantime, network, password, and data security is left up to the implementor and/or administrator. Different layers of security and some security mechanisms are discussed in[17, Chapter 11]. Actual implementation, however, is still up to the site.

Availability

Additional comments about minor problems and observations about implementing each of these facilities along with coding samples will be made available by request via e-mail. Also see the Usenix Web page in the author section.

Acknowledgements

I would like to thank Rob Kolstad for his valuable editorial assistance with this report. I would also like to thank Chris Demetriou for his valuable assistance in getting this report into its final form.

References

[1] Timothy A. Howes and Mark C. Smith, *Programming Directory-Enabled Applications with Lightweight Directory Access Protocol.* Macmillan Technical Publishing, Indianapolis, Indiana, 1997.

[2] Timothy A. Howes, *The Lightweight Directory Access Protocol: X.500 Lite.* Center for Information Technology Integration, University of Michigan, Ann Arbor, Michigan, 1995.

[3] Bryan Costales with Eric Allman, *Sendmail.* O'Reilly & Associates, Inc., http://www.oreilly.com/, second edition 1997.

[4] *Sendmail: src/READ_ME.* Sendmail Consortium, http://www.sendmail.org/, V8.[89].x.

[5] Booker Bense, *Using LDAP with sendmail.8.[89].x.* http://www.stanford.edu/∼bbense/Inst.html/, January 2000.

[6] *UMich LDAP-3.3.* University of Michigan, http://www.umich.edu/∼dirsvcs/ldap/, 1996.

[7] *The SLAPD and SLURPD Administrator's Guide, Release 3.3.* University of Michigan, http://www.umich.edu/∼dirsvcs/ldap/, 1996.

[8] *INN-2.1: README.perl_hook.* Internet Software Consortium, http://www.isc.org/, 1998.

[9] *INN-2.1: samples/filter_nnrpd.pl.* Internet Software Consortium, http://www.isc.org/, 1998.

[10] *INN-2.1: man(nnrp.access).* Internet Software Consortium, http://www.isc.org/, 1998.

[11] Frank Richter, *Web500GW.* http://www.tu-chemnitz.de/∼fri/web500gw/, V2.1b3, 1998.

[12] *LDAP-3.3: mail500/README.* University of Michigan, http://www.umich.edu/∼dirsvcs/ldap/, 1993.

[13] Don Libes, *Exploring Expect.* O'Reilly & Associates, Inc., 1st edition, 1994. [also, http://expect.nist.gov/]

[14] John D. Blair, *Samba - Integrating UNIX and Windows.* Specialized Systems Consultants, Inc., Seattle, Washington, 1998.

[15] *OpenLDAP-1.2.7.* OpenLDAP Foundation, http://www.openldap.org/, September 1999.

[16] Tim Howes and Mark Smith, *An LDAP URL Format.* RFC 1959, June 1996.

[17] Timothy A. Howes, Mark C. Smith and Gordon S. Good, *Understanding and Deploying LDAP Directory Services.* Macmillan Technical Publishing, Indianapolis, Indiana, 1999.

[18] Kartik Subbarao. *ldaptool.html* http://developer.netscape.com/viewsource/subbarao_ldap/subbarao_ldap.html/, September 1999.

[19] Ian Collier. *REXX-IMC-1.7.* http://users.comlab.ox.ac.uk/ian_collier/Rexx/rexximc.html/, February, 1999.

[20] Clayton Donley, Netscape Corporation, *perldap.* http://www.mozilla.org/directory/, 1999.

[21] Adrian Sun, *netatalk-1.4b2+asun-2.1.3*, ftp://ftp.cobaltnet.com/pub/users/asun/release/, [also: http://www.umich.edu/∼rsug/netatalk/], February 1999.

[22] Michal Trojnara <mtrojnar@ddc.daewoo.com.pl>, Adam Hernik <adas@infocentrum.com>, Pawel Krawczyk <kravietz@ceti.com.pl>, *stunnel-3.4a*. http://opensores.thebunker.net/pub/mirrors/stunnel/, December 1998.