CITI Technical Report 95–8

The Lightweight Directory Access Protocol: X.500 Lite

Timothy A. Howes
tim@umich.edu

ABSTRACT

This paper describes the Lightweight Directory Access Protocol (LDAP), which provides low-overhead access to the X.500 directory. LDAP includes a subset of full X.500 functionality. It runs directly over TCP and uses a simplified data representation for many protocol elements. These simplifications make LDAP clients smaller, faster, and easier to implement than full X.500 clients. Our freely available implementation of the protocol is also described. It includes an LDAP server and a client library that makes writing LDAP programs much easier.

July 27, 1995

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

The Lightweight Directory Access Protocol: X.500 Lite

Timothy A. Howes

July 27, 1995

1. Introduction

X.500, the OSI directory standard [1], defines a comprehensive directory service, including an information model, a namespace, a functional model, and an authentication framework. X.500 also defines the Directory Access Protocol (DAP) used by clients to access the directory. DAP is a full OSI protocol that contains extensive functionality, much of which is not used by most applications.

DAP is significantly more complicated than the more prevalent TCP/IP stack implementations and requires more code and computing horsepower to run. The size and complexity of DAP make it difficult to run on smaller machines such as the PC and Macintosh where TCP/IP functionality often comes bundled with the machine. When the DAP stack implementations are used, they typically require an involved customization process, which has limited the acceptance of X.500.

The Lightweight Directory Access Protocol (LDAP) was designed to remove some of the burden of X.500 access from directory clients, making the directory available to a wider variety of machines and applications. Building on similar ideas in the DAS [7] and DIXIE [4] protocols, LDAP runs directly over TCP/IP or other reliable transport. As we shall see, it simplifies many X.500 operations, leaving out little-used features and emulating some operations with others. LDAP uses simple string encodings for most attributes. The result is a low-overhead access method for the X.500 directory, suitable for use on virtually any platform.

Section 2 of this paper gives a quick introduction to X.500. Section 3 gives an overview of LDAP, describing the simplifications it makes to X.500. Section 4 summarizes the key advantages of the LDAP protocol. Section 5 briefly describes our implementation of LDAP, including our server and client library. Section 6 compares the performance of DAP and LDAP. Finally, Section 7 describes some work we are doing that builds on LDAP.

2. Overview of X.500

X.500 is the OSI directory service. X.500 defines the following components:

- An information model—determines the form and character of information in the directory.
- A namespace—allows the information to be referenced and organized.
- A functional model—determines what operations can be performed on the information.
- An authentication framework—allows information in the directory to be secured.

- A distributed operation model—determines how data is distributed and how operations are carried out.

The information model is centered around *entries*, which are composed of *attributes*. Each attribute has a *type* and one or more *values*. The type determines the attribute's *syntax*, which defines what kind of information is allowed in the values.

Which attributes are required and allowed in an entry are controlled by a special *objectClass* attribute in every entry. The values of this attribute identify the type of entry (e.g., person, organization, etc.). The type of entry determines which attributes are required, and which are optional. For example, the object class *person* requires the *surname* and *commonName* attributes, but *description*, *seeAlso*, and others are optional.

Entries are arranged in a tree structure and divided among servers in a geographical and organizational distribution. Entries are named according to their position in this hierarchy by a distinguished name (DN). Each component of the DN is called a relative distinguished name (RDN). *Alias* entries, which point to other entries, are allowed, circumventing the hierarchy. Figure 1 depicts the relationship between entries, attributes, and values and shows how entries are arranged into a tree.

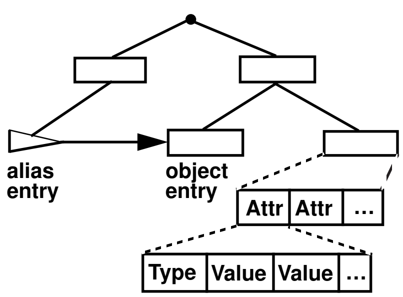

Figure 1. X.500 information model. The X.500 model is centered around entries composed of attributes that have a type and one or more values. Entries are organized in a tree structure. Alias entries can be used to build non-hierarchical relationships.

Functionally, X.500 defines operations in three areas: search and read, modify, and authenticate. In the first category, the *read* operation retrieves the attributes of an entry whose name is known. The *list* operation enumerates the children of a given entry. The *search* operation selects entries from a defined area of the tree based on some selection criteria known as a search filter. For each matching entry, a requested set of attributes (with or without values) is returned. The searched entries can span a single entry, an entry's children, or an entire subtree. Alias entries can be followed automatically during a search, even if they cross server boundaries.

In the second category, X.500 defines four operations for modifying the directory. The *modify* operation is used to change existing entries. It allows attributes and values to be added and deleted. The *add* and *delete* operations are used to insert and remove entries from the directory. The *modify RDN* operation is used to change the name of an entry.

The final category defines a *bind* operation, allowing a client to initiate a session and prove its identity to the directory. Several authentication methods are supported, from simple clear-text password to public key-based authentication. The *unbind* operation is used to terminate a directory session. An *abandon* operation is also defined, allowing an operation in progress to be canceled.

Each X.500 operation and result can be *signed* to ensure its integrity. Signing is done using the originating client's or server's public key. The signed request or result is carried end-to-end in the protocol, allowing integrity to be checked at every step. This guards against connection hijacking or modification by intermediate servers. *Service controls* can be specified that determine information such as how an operation will be carried out, whether aliases should be dereferenced, the maximum number of entries to return, and the maximum amount of time to spend on an operation.

In X.500, the directory is distributed among many servers (called DSAs for Directory

System Agent). No matter which server a client connects to, it sees the same view of the directory. If a server is unable to answer a client's request, it can either *chain* the request to another server, or *refer* the client to the server.

3. Overview of LDAP

LDAP assumes the same information model and namespace as X.500. It is also client-server based, with one important difference: there are no referrals returned in LDAP. An LDAP server must return only results or errors to a client. If referrals are involved, the LDAP server is responsible for chasing them down. This model is depicted in Figure 2, though the intermediate server shown is not required (i.e., an implementation could choose to have its DSA speak "native" LDAP).

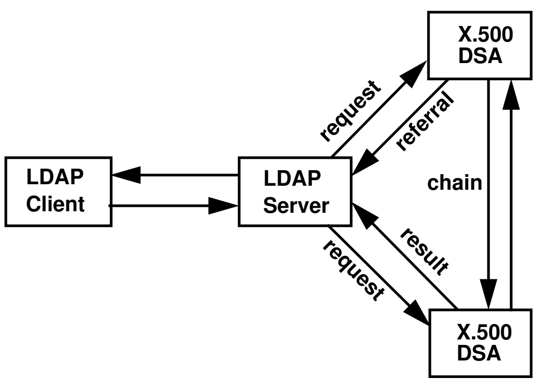

Figure 2. Relationship between LDAP and X.500. The LDAP client-server model includes an LDAP server translating LDAP requests into X.500 requests, chasing X.500 referrals, and returning results to the client.

The LDAP functional model is a subset of the X.500 model. LDAP supports the following operations: search, add, delete, modify, modify RDN, bind, unbind, and abandon. The search operation is similar to its DAP counterpart. A base object and scope are specified, determining which portion of the tree to search. A filter specifies the entries within the scope to select. The LDAP search filter offers the same functionality as the one in DAP but is encoded in a simpler form. The time and size limit service controls are included directly in the search request. (They are not included with the other operations.) The *searchAliases* search control and *dereferenceAliases* service control are combined in a single *derefAliases* parameter in the LDAP search. The ASN.1 [11] definition of the LDAP search request is shown in Figure 3.

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
 baseObject  LDAPDN,
 scope       ENUMERATED {
  baseObject   (0),
  singleLevel  (1),
  wholeSubtree (2)
 },
 derefAliasesENUMERATED   {
    neverDerefAliases     (0),
    derefInSearching      (1),
    derefFindingBaseObj   (2),
    alwaysDerefAliases    (3)
 },
 sizeLimit     INTEGER (0 .. MaxInt),
 timeLimit     INTEGER (0 .. MaxInt),
 attrsOnly     BOOLEAN,
 filter        Filter,
 attributes    SEQUENCE OF AttributeType
}
Filter ::= CHOICE {
   and            [0] SET OF Filter,
   or             [1] SET OF Filter,
   not            [2] Filter,
   equalityMatch  [3] AttributeValueAssertion,
   substrings     [4] SubstringFilter,
   greaterOrEqual [5] AttributeValueAssertion,
   lessOrEqual    [6] AttributeValueAssertion,
   present        [7] AttributeType,
   approxMatch    [8] AttributeValueAssertion
}
```

Figure 3. ASN.1 for the LDAP search operation. The LDAP search operation offers similar functionality to DAP search. It combines search parameters and service controls and simplifies the filter encoding.

The *LDAPDN* and *AttributeType* components of the search are encoded as simple character strings using the formats defined in RFC 1779 [5] and RFC 1778 [2], respectively, rather than the highly structured encoding used by X.500. Similarly, the value in an *AttributeValueAssertion* is encoded as a primitive OCTETSTRING, not a more structured ASN.1 type. The structure is reflected in the syntax of the encoded string, not in the encoding itself.

The results of an LDAP search are sent back to the client one at a time, in separate *searchEntry* packets. This sequence of entries is terminated by a final *searchResult* packet that contains the result of the search (e.g., success, a size or time limit was exceeded, etc.). Having a final terminator packet allows clients and servers to *stream* results more easily, handling one entry at a time. This is especially useful in memory-constrained environments where holding the collection of all entries from a large search is not possible.

The X.500 list and read operations are not included in LDAP. Instead, they are emulated with the LDAP search operation. Read is emulated by a base object search of the entry to read, with a filter testing for the existence of the *objectClass* attribute. Every entry is required to have an object class and must match this filter. List is emulated by a one level search of the entry to list, also with a filter testing for the existence of the *objectClass* attribute. If the ability to distinguish alias children from other children (a feature provided by X.500 list) is desired, the *objectClass* attribute can be retrieved and examined for a value of *alias*.

The LDAP modify operation also differs slightly from its DAP counterpart. In DAP, four kinds of changes can be made: entire attributes can be added or deleted; individual values can be added or deleted. These capabilities require a client to read an entry before attempting a modify (e.g., when adding a value, to discover whether an *add attribute* or *add value* is required).

In LDAP, we simplified the semantics of modify by supporting three operations: add values; delete values; and replace values. If a request is made to add values to an attribute that does not exist in the entry, the attribute is created automatically. If a request is made to delete the last value of an attribute, the entire attribute is deleted. An attribute can also be deleted by specifying a *delete values* operation without specifying any values. Finally, the *replace values* construct is used to make an attribute contain the given values after the modify. The LDAP server takes care of translating the replace request into the necessary sequence of modify, add, and delete operations required by X.500.

The LDAP bind operation supports a subset of X.500 bind functionality. It allows only simple authentication, consisting of a cleartext password, and Kerberos version 4 authentication [6], which translates into an X.500 external authentication method. The LDAP bind operation includes a choice of credentials, allowing for future expansion of available authentication methods.

The DAP unbind, abandon, modify RDN, add and delete operations are virtually identical to their DAP counterparts.

4. Key Advantages

LDAP has four key advantages over DAP. First, it runs directly over TCP (or other reliable transport, in theory), eliminating much of the connection set-up and packet-handling overhead of the OSI session and presentation layers required by DAP. In addition, the near universal availability of TCP/IP implementations means that LDAP can run on most systems "out of the box."

Second, LDAP simplifies the X.500 functional model in two ways. It leaves out the read and list operations, emulating them via the search operation. It also leaves out some of the more esoteric and less-often-used service controls and security features of full X.500 (e.g., the ability to sign operations). This simplifies LDAP implementations.

Third, though X.500 and LDAP both describe and encode protocol elements using ASN.1 and BER [12], LDAP uses string encodings for distinguished names and data elements. X.500 uses a complex and highly-structured encoding even for simple data elements; LDAP data elements are string types. This encoding is a big win for distinguished names, which have considerable structure leading to encoding/decoding complexity and size. LDAP relegates the knowledge of a value's syntax to the application program rather than lower-level protocol routines.

Finally, LDAP frees clients from the burden of chasing referrals. The LDAP server is responsible for chasing down any referrals returned by X.500, returning either results or errors to the client. Clients assume a single connection model in which X.500 appears as a single logical directory.

5. Implementation

In setting out to implement LDAP we had three goals in mind:

- provide a freely available reference implementation of the protocol;
- enable the development of LDAP clients on a wide variety of platforms; and
- solve the problem of providing access to our campus X.500 directory.

In addition, we have found our implementation has been incorporated into a number of vendor offerings, increasing the availability of LDAP products.

Our LDAP implementation has three main components: a server, a client library, and various clients. Our LDAP server, *ldapd*, is based on the popular ISO Development Environment (ISODE) package. We use the ISODE OSI stack implementation and DAP client library to access X.500. The *ldapd* server supports connections to multiple X.500 servers, providing efficient handling of referrals.

The *ldapd* server can be run as a UNIX stand-alone daemon or from *inetd*, the UNIX Internet protocol daemon. It accepts connections from LDAP clients, forking off a copy of itself to handle each connection. It also supports connectionless LDAP (CLDAP) [10], a version of LDAP that runs over UDP or other connectionless transport. CLDAP is useful in applications where speed is paramount, the information desired is small, and the connection setup overhead of LDAP is too large.

Key to the success of our LDAP implementation has been *libldap*, the LDAP client library. The *libldap* library gives programmers a simple yet powerful C Language API for accessing the X.500 directory through LDAP. The library is self-contained, including the necessary ASN.1/BER routines for producing and reading LDAP protocol elements. It contains routines to begin and end sessions with the directory, perform searches and other operations, and parse and display the results obtained from the directory. Figure 4 is a C code fragment showing a simple use of *libldap*. It illustrates the synchronous interface provided by *libldap*. Asynchronous routines are also provided.

```
#include <ldap.h>
LDAP        *ld;
LDAPMessage  *e, *r;
char        *a, *dn;
/* open a connection and authenticate */
if ((ld = ldap_open("hostname", LDAP_PORT))
 == NULL)
     fail();
if (ldap_simple_bind_s(ld, NULL, NULL) !=
 LDAP_SUCCESS)
     fail();
/* search for entries, return all attrs */
if (ldap_search_s(ld, "c=US", LDAP_SCOPE_ONELEVEL,
 "o=*michigan*", NULL, 0, &r) != LDAP_SUCCESS)
     fail();
/* step through each entry returned */
for (e = ldap_first_entry(ld, r); e != NULL;
  e = ldap_next_entry(ld, e)) {
     /* get and print the entry name */
     dn = ldap_getdn(ld, e);
     printf("entry: %s\n", dn);
     free(dn);
     /* step through each attribute */
     for (a = ldap_first_attribute(ld, e);
         a != NULL;
         a = ldap_next_attribute(ld, e, a)) {
             printf("attr: %s\n", a);
             /* get and print vals */
             v = ldap_get_values(ld, e, a);
             for (i = 0; v[i] != NULL; i++) {
                 printf("val: %s\n", v[i]);
             }
             ldap_value_free(v);
     }
}
```

Figure 4. Sample *libldap* code. This code fragment searches for and retrieves entries from the directory. The entries are then stepped through and each value of each attribute is printed. If the attribute names retrieved are known, `ldap_get_values()` can be called with the names directly.

In addition to the basic operations shown in Figure 4, *libldap* contains routines to assist LDAP application developers in a variety of ways. There are *display template* routines which provide a standardized way of displaying entries. The display format is governed by a configuration file that tells which attributes to display for entries of a particular object class and how to display them. By using these routines, no code changes are necessary for an application to change how entries are displayed, add a new attribute to the display, etc.

Also provided are routines to assist in the construction of search filters. Often, different filters need to be constructed based on user input. For example, in a simple look-up application if a user types in a number, one might want to perform a search for entries with a phone number (home, work, fax, or pager) matching all or part of the number. If an alphabetic string is input, a search by name is more appropriate. If an exact match search yields no results, a less restrictive approximate search might be tried. The *get filter* routines automate the process of creating these filters. The filters produced are specified in a configuration file via regular expressions that are matched against user input.

Many LDAP client applications have been developed by us and others on the Internet. Some of the more interesting applications include maX.500, waX.500 and xax500, GUI clients for the Macintosh, MS Windows, and X Windows, respectively; go500gw, a gopher to X.500 gateway; web500gw, a World Wide Web to X.500 gateway; and mail500 and fax500, RFC 822-based X.500-capable mailers. Work is ongoing on other applications as well.

6. Performance

The performance of LDAP is satisfactory for most applications. In this section, we compare the performance of DAP and LDAP in four areas: response time to queries; the size of queries; PDU encoding speed; and the size and complexity of client-side implementations. For these comparisons, we used our LDAP implementation and the ISODE DAP implementation. The same DSA was used for all query measurements, providing a baseline for comparison.

Table 1 shows the performance of a range of typical DAP and LDAP queries. The tests were conducted on a dedicated machine running the DAP and LDAP clients, the LDAP server, and the DSA. As can be seen in the table, the delay introduced by LDAP is minimal. This delay could be eliminated altogether by a native DSA implementation, eliminating the intermediate encoding, decoding, and protocol translation.

Table 1. Comparison of DAP and LDAP query times. Searches were performed using the same DSA, with a "hot" cache of entries. Times are in milliseconds.

Query	DAP	LDAP
Unauthenticated bind	30	68
Authenticated bind	34	56
Simple search (one entry)	32	41
Simple search (50 entries)	293	353

Table 2 shows the size of the queries and results given in Table 1. It shows that LDAP queries are substantially smaller than equivalent DAP queries. The savings are due primarily to the simplified DN and value encodings. Query sizes are also reduced by the absence of service controls in every operation.

Table 2. Comparison of DAP and LDAP query sizes. LDAP queries are significantly smaller than their DAP counterparts. Query sizes are in bytes.

Query	DAP	LDAP
Unauthenticated bind	192	14
Authenticated bind	409	138
Simple search request	237	105
Single search result	547	355

Tables 3 and 4 show the time to decode and encode a range of typical DAP and LDAP PDUs. They show that LDAP has a modest performance advantage for simple PDUs and a substantial advantage for complex PDUs, especially those containing many distinguished names where the LDAP string representation is a big win.

Table 3. Comparison of DAP and LDAP decoding times. LDAP protocol elements are easier to decode, especially for complex PDUs. The complex PDU contained an attribute with over 600 DNs. About half of the DAP decoding time was spent in a duplicate check, to ensure that an attribute has only one of each value.

PDU Complexity	DAP	LDAP
Simple	550	110
Medium	7,925	714
Complex	38,393	2,702

Table 4. Comparison of DAP and LDAP encoding times. LDAP protocol elements are encoded more efficiently, especially for complex PDUs.

PDU Complexity	DAP	LDAP
Simple	24	6
Medium	1,084	324
Complex	2656	989

Finally, we compare implementation size and code complexity). Such a comparison is anecdotal at best, given the wide range of programmer skills and goals used in producing the implementations. However, some conclusions favorable to LDAP can be drawn from the overwhelming advantage it has in this area, as shown in Table 5.

The Directory Enquiries client was chosen for the size comparison. It can be compiled to use either DAP or LDAP for X.500 communication. The code complexity of the ISODE DAP and our LDAP client libraries were also compared. We used two complexity measures. The first, a count of the number of semi-colons, approximates the number of statements. The second, a count of the number of "if" statements, approximates the number of code paths. In computing both of these metrics, an effort was made to include only those portions of code required to access X.500.

Table 5. Comparison of DAP and LDAP implementation complexity. The DE client, which can be built using either DAP or LDAP, is used to compare implementation size. Semicolon count, which approximates the number of statements, and "if" statement count, which approximates the number of code paths are another measure of complexity. The comparison was between ISODE-8.0 and our LDAP implementation.

Metric	DAP	LDAP
Total size (DE)	1,484,568	334,552
Text	958,464	221,184
Data	385,024	73,728
BSS	141,080	39,640
Semicolon count	46,746	1,989
If count	9369	568

7. Future Work

LDAP has succeeded in making X.500 more accessible and is largely responsible for a substantial increase in X.500 client development. Despite this success, X.500 deployment on the Internet remains disappointing. One reason for this is the heavyweight nature of X.500 servers; to take advantage of the proliferation of LDAP clients to access local data, a site must first bring up a full X.500 service. To address this problem we are developing a *stand-alone* LDAP server called *slapd*. *Slapd* exports the same LDAP functionality described above but is backended by its own local database, not by X.500.

To prevent stand-alone LDAP servers from being isolated from the rest of the X.500 world, we have made a compatible extension to LDAP that allows the return of referrals to the client. This adds some complexity on the client side to follow the referrals, but in return we gain simplicity in the server.

The 1993 version of the X.500 standard includes many features missing from 1988 X.500, on which LDAP is based. Among the new features are access control, replication, schema management, and various DAP extensions. A new version of LDAP is under development by the Internet Engineering Task Force that will incorporate some of these features, as well as address some security concerns with the present version of LDAP, such as its lack of strong authentication and integrity insurance capability.

The DAP extensions include the ability to retrieve search results a "page" at a time, specify a byte limit on the size of an attribute to return, treat the attributes of a DN as part of the entry during a search, and more. The security features being considered include strong (public key-based) authentication, and signing of operations.

Finally, with the growing popularity of the World Wide Web, we see interesting and exciting possibilities for merging the two technologies. Work has already begun on defining a URL format for LDAP [3], and a URL-valued attribute for X.500 [8].

8. Summary

The Lightweight Directory Access Protocol provides a low-overhead method of accessing the X.500 directory. It runs directly over TCP, and makes several simplifications to full X.500 DAP, leaving out many of the lesser-used features. LDAP uses primitive string encodings for most data elements, making it more efficient and easier to implement than DAP. We have developed a freely available reference implementation of LDAP which has been ported to several platforms, including UNIX, VMS, PC, and Macintosh. Our intermediate-server-based implementation introduces little delay over full DAP, produces smaller protocol exchanges, and results in smaller and less complex clients. Our implementation is freely available:

`ftp://i7465726d696e61746f722e7273e697464e756d696368e656475z.oszar.com/ldap/ldap.tar.Z`

There is also an LDAP discussion list joinable by sending email to:

`ldap-request@umich.edu`

9. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. NCR-9416667. LDAP was developed in collaboration with Steve Kille, Wengyik Yeong, and Colin Robbins, along with help from members of the Internet Engineering Task Force. My colleague Mark C. Smith deserves much of the credit for the LDAP implementation described in this paper. Many thanks also to Peter Honeyman for his ever-valuable review.

References

1. "The Directory: Overview of Concepts, Models and Service," CCITT Recommendation X.500, 1988.
2. T. Howes, S. Kille, W. Yeong, and C. Robbins, "The String Representation of Standard Attribute Syntaxes," RFC 1778, March 1995.
3. T. Howes and M. Smith, "An LDAP URL Format," Internet Draft draft-ietf-asid-dap-format-00.txt, March 1995.
4. T. Howes, M. Smith and B. Beecher. "DIXIE Protocol Specification," RFC 1249, August 1991.
5. S. Kille, "A String Representation of Distinguished Names," RFC 1779, March 1995.
6. S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, "Kerberos Authentication and Authorization System," MIT Project Athena Documentation Section E.2.1, December 1987.
7. M. Rose, "Directory Assistance Service," RFC 1202, February 1991.
8. M. Smith, "Definition of an X.500 Attribute Type and Object Class to Hold Uniform Resource Identifiers (URIs)," Internet Draft draft-ietf-asid-x500-url-01.txt, March 1995.
9. W. Yeong, T. Howes, and S. Kille, "Lightweight Directory Access Protocol," RFC 1777, March 1995.
10. A. Young, "Connectionless Lightweight Directory Access Protocol," Internet Draft draft-ietf-osids-cldap-02.txt, April 1995.

11. Specification of Abstract Syntax Notation One (ASN.1), CCITT Recommendation X.208, 1988.

12. Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), CCITT Recommendation X.209, 1988.

Author Information

Tim Howes is a Senior Systems Research Programmer for the University of Michigan's Information Technology Division. He received a B.S.E. in Aerospace Engineering, a M.S.E. in Computer Engineering from U-M, and is completing a Ph.D. in Computer Science. He is currently project director and principal investigator for the NSF-sponsored WINX project, and in charge of directory service development and deployment at U-M. He is the primary architect and implementor of the U-M LDAP directory package, the DIXIE system, the GDA X.500 DSA, and a major developer of the QUIPU X.500 implementation. He is author or co-author of several papers and RFCs, including RFC 1777 and RFC 1778 defining the LDAP protocol. He is chair of the IETF Access, Searching, and Indexing of Directories working group, and an active member of the ACM and IEEE.