Measurement and Analysis of LDAP Performance

Xin Wang, Henning Schulzrine
Department of Computer Science
Columbia University
New York, NY 10027
{xinwang, hgs}@cs.columbia.edu

Dilip Kandlur, Dinesh Verma
Network Systems Department
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{kandlur, dverma}@watson.ibm.com

ABSTRACT

The Lightweight Directory Access Protocol (LDAP) is being used for an increasing number of distributed directory applications. We describe a tool to analyze the performance of LDAP directories. Using the tool, we study the performance of a LDAP directory under a variety of access patterns. For the purpose of these experiments, we use a LDAP schema that has been proposed for the administration of *Service Level Specifications* (SLSs) in a differentiated services network. In addition, individual modules in the server and client code are instrumented to obtain a detailed profile of their contributions to the overall system latency and throughput. We discuss the importance of the factors in determining scalability, namely front-end versus back-end processes, CPU capability, and available memory. Under our normal operating conditions, a LDAP directory with 10,000 entries is found to have a response time of 8 ms, and a maximum throughput of 140 search requests per second. Out of the total response latency, approximately 3 ms is associated with connection management, 1.8 ms with retrieval of entries from the database, and 3.2 ms with front-end processing. At high loads, the connection management latency increases sharply to dominate the response in most cases. The Nagle algorithm is found to introduce a very large additional latency, and it appears beneficial to disable it in the LDAP server. The CPU capability is found to be significant in limiting the performance of the LDAP server, and for larger directories, which cannot be kept in memory, data transfer from the disk also plays a major role. The scaling of server performance with the number of directory entries is determined by the increase in back-end search latency, and scaling with directory entry size is limited by the front-end encoding of search results, and, for out-of-memory directories, by the disk access latency. We investigate different mechanisms to improve the server performance.

1. INTRODUCTION

The Lightweight Directory Access Protocol (LDAP) is being used for an increasing number of directory applications. Applications include personnel databases for administration, tracking schedules [1], address translation databases for IP telephony, network databases for storing network configuration information and service policy rules [2, 13, 14], and storage of authentication rules [3, 4]. In many of these cases, such as using LDAP directories for storage of personnel information and authentication rules, the data is relatively static, so that caching can be used to improve performance. In some situations, the database information needs to be updated frequently. For example, in IP telephony, every time a subscriber uses a different terminal or is at a different location, his account information may need to be updated. Despite the growing importance of LDAP services, there has been little work on how LDAP servers behave under different workloads, and in different operating environments. In particular, the performance of LDAP in a dynamic environment with frequent searches has not been looked at closely.

In this paper, we report on the development of a tool to benchmark LDAP server performance, and analyze results derived using this tool. In addition, we have instrumented detailed profiling at the server and LDAP client API codes. These results include the contribution of various system components to the overall performance in terms of latency and throughput, the scaling of performance with directory size, entry size, and session re-use, and the importance of various factors in determining scalability. We also investigate modifications and usage patterns which lead to an improvement in server performance.

Given the growing use of LDAP in applications, it is useful and convenient to carry out the performance experiments using data based on an existing LDAP schema proposed for a real directory application. In this work, we use a schema proposed in [13, 14] for the administration of *Service Level Specifications* (SLSs), which are used to configure networks for supporting different levels of services. In this application, it is envisioned that the administrative policies embodied in the LDAP schema will be stored on directories and downloaded to devices such as hosts, routers, policy servers, proxies, etc. If the SLS is allowed to be dynamically negotiated [7, 8], the LDAP service must deal with frequent directory queries. In these respects, this application is representative of many current or proposed LDAP applications [2, 3, 4]. The results reported in this work should be generally applicable to many of the applications cited earlier; aspects of the work that are specific to SLS administration will be pointed out where appropriate.

The rest of this paper is organized as follows. In Section 2, we first provide a general background on the LDAP directory service, and then provide a very brief introduction to differentiated service networks and service level Specifications, as well as the LDAP schema proposed for this application. The experimental set-up is discussed in Section 3, followed by a discussion of the test methodology in Section 4. Experiments are described, and the results are presented and analyzed in Section 5, and related work is presented in Section 6. Finally, we summarize our results and present some conclusions

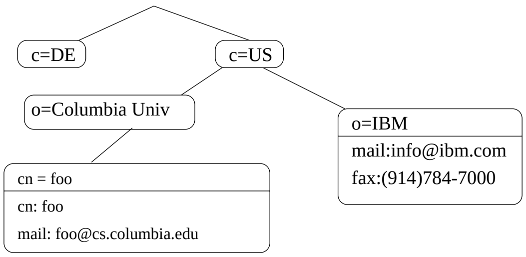

Figure 1: An example of organization of data in a LDAP directory

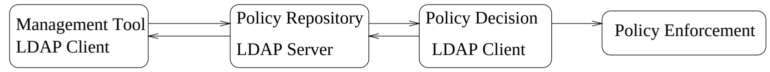

Figure 2: An architecture for network QoS control using LDAP

in Section 7.

2. BACKGROUND

In this section, we first provide a brief introduction to the LDAP directory service. We then provide a background on the use of LDAP in administration of differentiated services networks. In this context, we also describe the LDAP directory structure used in our experiments.

2.1 The LDAP Directory Service

A directory service is a simplified database. Typically, it does not have the database mechanisms to support roll-back of transactions. Directories allow both read and write operations, but are intended primarily for high-volume, efficient read operations by clients.

LDAP is a distributed directory service protocol. LDAP is based on a client-server model and runs over TCP/IP. It can be used to access stand-alone directory servers or X.500 directories. Today, LDAPv2 is an Internet standard as defined by the IETF standards process. The standards document, RFC 1777 [9], dates back to March 1995. A newer specification, LDAPv3 [10], is currently a draft in review that is expected to become a new standard soon.

Information is stored in a LDAP directory in the form of entries arranged in a hierarchical tree-like structure (Figure 1). An LDAP entry is a collection of *attributes*, for example, an entry corresponding to a person may have as its attributes the name of the person, organization, email-address. Each attribute has a *type*, which is an identifying mnemonic (for example, the email attribute may have type “mail”) and the attribute takes one or more *values* (the email attribute might have “foo@cs.columbia.edu” as a value).

LDAP defines operations for querying and updating the directory. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria.

2.2 Using LDAP for SLS Administration

As mentioned earlier, although we assume a LDAP directory intended for storage of SLS policies, most of the experimental results presented in this work apply to LDAP services in general, and a detailed understanding of differentiated service networks and Service Level Specifications is not required to follow the rest of this paper. However, a brief background may be of interest to some readers.

2.2.1 Service Level Specifications

The current Internet operates on a best-effort basis, in which all packets are treated equally. Recently, there has been much interest in network service models with mechanisms to provide multiple service levels to users. The two main approaches under discussion are the integrated service model, which supports QoS levels by allowing per-flow resource reservation using RSVP [5] signaling, and the differentiated service model [6, 7, 8], which provides multiple service classes which are served using different per-hop behaviors. In either model, the network provider negotiates a service level specification with a customer, defining aspects of network behavior such as the type of service user packets will receive, and the constraints the user traffic must adhere to. The SLS may be dynamically re-negotiated, based on changes in the customer requirements or network conditions.

The network access points and internal routers implement the classification, resource control, and administrative policies associated with SLSs. Researchers in the DiffServ community have proposed storing these policies in a central or distributed policy repository administered and accessed using a directory service such as LDAP [13, 14, 15]. In the proposed scenario, the policy repository is updated when the network provider negotiates new SLSs, or re-negotiates existing contracts, and also when the policies need to reflect changes in network topology or traffic levels. Network elements frequently access the policy database, and download the current set of rules according to which customer traffic is served.

In addition, the network provider provisions the network in order to provide the service contracted to customers. The provisioning is physical (adding or removing network elements) and logical (partitioning or configuring network elements). The network configuration information may be maintained in LDAP directories, and downloaded periodically by routers. This allows the network provider to adjust configurations (for example, buffer space, or packet drop precedences) with a finer granularity in response to network usage feedback.

2.2.2 Architecture of Network QoS Control Using LDAP

A preliminary schema using LDAP for configuration of DiffServ networks has been proposed in [13]. The various aspects of a service, such as the traffic profile the user traffic must conform to in order to receive the service, and the forwarding rules for conforming traffic, are captured in a set of policies. The generic architecture that is envisioned consists of a management tool, a policy repository, a policy decision entity, and a policy enforcement entity. Figure 2 shows the functional relations between these different entities, and it does not restrict where these functional entities should be located.

In the context of the service environment under consideration, the management tools are used by the network administrator to populate and maintain the LDAP directory with policies. Management tools may or may not reside on the same host as the directory server. Enforcement entities apply policy rules.

A decision entity and enforcement entity are usually assumed to reside at each edge device, or network access point. The edge device is referred to by its location and would most likely be placed at the access point between a local subnet and the backbone network, or at the boundary between backbone networks of two service providers. The decision entity downloads policy rules from the repository, through a LDAP client. The enforcement entity queries rules from the decision entity and carries out packet handling and monitoring functions. The decision entity may either download the entire policy repository all at once, or may query the directory when needed - for instance, when triggered by events such as an RSVP message or an IP packet bearing a TCP connect request.

A customer attaches to the network at one or more interfaces belonging to an edge device. Each interface is identified by an IP address. At each interface, one or more policies may be defined, and customer packets are monitored and processed according to these policies. Each policy is associated with a service level which defines actions on the part of network elements in handling customer packets. A policy may be applied on the basis of source/destination IP addresses, transport protocols, source/destination ports, and other parameters such as default port, URL's, etc.

Policy rules are stored in the LDAP directory as SLSPolicyRules objects (derived from the *Policy* class described in [13]). SLSPolicyRules objects may have attributes specifying the policy name, priority level of the rule, and the network interfaces to which the rule may be applied, as well as references to objects which specify the traffic profile, period of validity of the rule, type of RSVP service or DiffServ action, etc.

At initialization, the edge device identifies its interface addresses. It determines the set of policies required for these interfaces, and downloads the corresponding classification policy rules from the LDAP server, as well as the service specifications referred by the policies. Subsequently, the edge device may poll the server periodically to learn of modifications to the directory, and download its set of policy rules if the directory is modified. If asynchronous mode operations are supported by the directory service, the downloading of policy rules could also be triggered upon changes in the policy rules.

2.2.3 LDAP Directory Structure Used in the Experiments

The LDAP directory structure used in our tests is a simplified version of the directory used to develop the LDAP schema for supporting SLS [13, 14, 15], and is shown in Figure 3. Each *Customer* entry has a set of associated *Interface* entries. The *Policy* entry directly under the *Customer* specifies policy rules common to multiple interfaces belonging to the customer, while the *Policy* entry for each *Interface* specifies the policy rules specific to customer traffic at that Interface. In general, the *Policy* entry refers to one or more of the *Service* entries in the directory to specify the service to be received by the corresponding traffic. The other entries shown in the LDAP directory include *Channel* and *Pacer* entries. A channel is a virtual pipe between an ingress edge-device and an egress edge-device. A pacer is the abstraction that limits the total amount of traffic that can be sent out into the backbone network at an access-point.

3. EXPERIMENTAL SETUP

In this section we describe our experimental testbed, including the hardware we use, the LDAP server software structure, the LDAP client load generation and the benchmarking setup.

3.1 Hardware

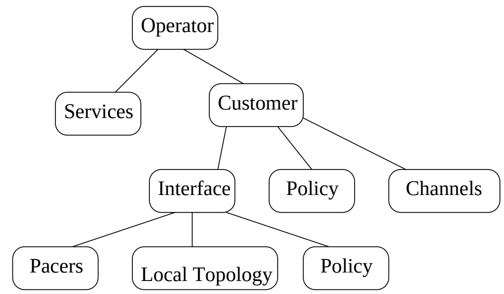

Figure 3: LDAP tree structure in tests

The LDAP server ran on a dual-processor Ultra-2 machine equipped with two 200 MHz Sun, UltraSPARC CPUs, 256 MB main memory. The LDAP server process was bound to one of the two CPUs. The LDAP clients ran on a couple of Sun Ultra 1 models with 170 MHz CPU, 128 MB main memory, and one Sun ultra 10 machine with 299 MHz CPU and 256 MB main memory. The server and clients were connected via 10 Mb/s Ethernet.

3.2 LDAP Server

There are a number of commercial LDAP servers, including Netscape Directory Server, and Novell LDAP Services. We chose OpenLDAP 1.2 [11]. OpenLDAP is a complete open source suite of client and server applications derived from University of Michigan LDAP v3.3. The main reasons for our using OpenLDAP is its open source model, and its rapidly increasing user population. The open source model allowed us to perform detailed profiling of individual server modules and examine some modifications of the basic implementation instead of treating the server as a black box. The server is based on a stand-alone LDAP daemon (*slapd*) for directory service. Replicated service is also supported through a UNIX daemon *slurpd*. In this work, the goal is to study the performance and scalability of the server, and we restrict the LDAP clients to connect to one slapd. Slapd consists of two distinct parts: a front end that handles protocol communication with LDAP clients; and a backend that handles database operations. Slapd comes with three different backend databases to choose from. They are LDBM, a high-performance disk-based database; SHELL, a database interface to arbitrary UNIX commands or shell scripts; and PASSWD, a simple password file database. The LDBM backend relies on a low-level hash or B-tree package for its underlying database. In this work, we used an LDBM backend, namely the Berkeley DB version 2.4.14 package [12] hash database.

LDBM has two important configuration parameters: *cachesize*, the size in entries of an in-memory cache that holds LDAP directory entries, and *dbcachesize*, the size in bytes of the in-memory cache associated with each open index file. In our experiments, *dbcachesize* was set equal to 10 MB, sufficiently large to keep all index files in-memory. The *cachesize* varied according to specific experiments.

3.3 LDAP Client

The overall client-server architecture used in our experiments is shown in Fig. 4. A collection of client machines are connected to a server machine. There can be more than one LDAP process running on a client or server machine. The client machines used had sufficient CPU capability that the delay at client sites could be ignored in measuring server performance.

A Bench Master process coordinates the client processes and gener-

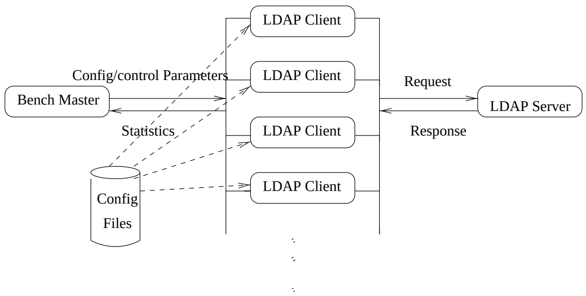

Figure 4: LDAP benchmarking testbed architecture

ates an overall performance report. The setup parameters are defined in configuration files. The Bench Master constructs the command-line arguments for LDAP client processes based on the configuration files. The Bench Master is also responsible for spawning the LDAP clients remotely on the designated machines. Each of the LDAP clients reads the command line and starts up communication with the Bench Master. After all the LDAP clients have been initialized, the Bench Master instructs the LDAP clients to commence the benchmarking. It then waits for the end of the experiment when it receives all the statistics from all the LDAP clients. Each LDAP client queries the LDAP directory periodically, with the query starting time for each client randomized to prevent synchronization. In most of our experiments, the workload of the LDAP server was changed by varying the query interval of the LDAP clients. The Bench Master organizes the data from the clients into the benchmark report.

4. TEST METHODOLOGY

Common LDAP operations are *modify*, *add*, *delete*, *compare* and *search*. In the directory application considered for our experiments, the service specifications should remain relatively static during normal operation, while the policies defined by customer-provider SLSs would be updated much more often, as customers negotiate new SLSs and re-negotiate old SLSs. *Search* operations are therfore likely to dominate the server load. In general, this is true for most LDAP applications. Accordingly, for most of our experiments the server workload consisted of *search* requests for downloading of policy rules (SLSPolicyRules objects) from the LDAP directory (Fig. 2).

The search filter for the search operation was constructed from the Interface address of interest, and the corresponding *Policy* object. The default entry size for most experiments was 488 bytes, and the default directory size was 10,000 entries.

A simple LDAP search involves a sequence of 4 operations: *ldap_open*, *ldap_bind*, one or more *ldap_search* operations, and *ldap_unbind* (Figure 5). *ldap_open* initializes the LDAP library, opens a connection to the directory server, and returns a session handle for future use. The *ldap_bind* operation is responsible for client authentication. The bind operation allows a client to identify itself to the directory server by using a Distinguished Name and some authentication credentials (a password or other information). LDAP supports a variety of authentication methods. In our experiments, password authentication was used. When a bind operation is successfully completed, the directory server remembers the new identity until another bind is done or the LDAP session is terminated by calling *ldap_unbind*. The identity is used by the server to make decisions about what kind of changes can be made to the directory. The *ldap_search* operation

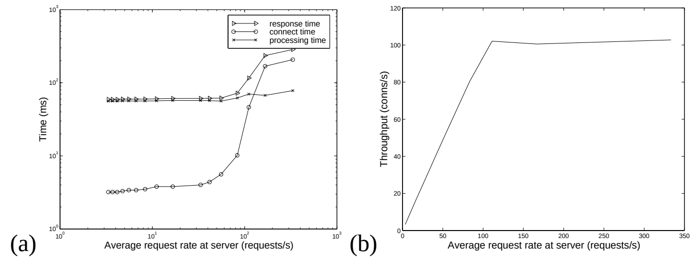

Figure 6: Average connection time, processing time and response time (a) and average server throughput (b) shown as a function of the average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes.

initiates a LDAP search by specifying the criteria that entries fitting in the associated filter could be returned. The search filters were randomized in our experiments to avoid search results being cached. Finally, an LDAP session is disposed of by using *ldap_unbind*.

Referring to Fig. 5, we define the following latency measures: the connect time is defined as the time from the sending of *ldap_open* request until *ldap_bind* operation is successful. The processing time is defined as the time required for the *ldap_search* operation as well as the data transfer time for the retrieved results to be returned to the clients. The total time required for an LDAP search operation, from *ldap_open* to *ldap_unbind*, is defined as the response time. The measures that reflect the performance of the LDAP service are the connect, processing and response latencies, and the throughput of the server, represented by the average number of requests served per second.

In our experiments, each search operation involved all four of the above steps, *ldap_open*, *ldap_bind*, *ldap_search*, and *ldap_unbind*. In a real application, a client performing multiple searches may prefer to leave the connection open and only do an unbind at the end. In this sense, the total response time data in the experiments represents a worst-case scenario. In Section 5.4, we consider the effect of leaving the connection open for multiple searches or for the duration of the experiment on performance.

In addition to the *search* experiments, the performance of the LDAP server for database updates is studied in Section 5.5, using a workload consisting of *ldap_add* requests for adding SLSPolicyRules objects. In this case, the client must construct a new entry with a set of attributes before calling the *ldap_add* routine.

5. RESULT ANALYSIS

Our experiments have three main purposes: identify the contributions of various system components towards the overall LDAP performance; suggest measures to improve performance; and study the limits of LDAP performance, and what determines these limits. We organize the experimental results as follows. The overall performance with respect to throughput and latency is introduced in Section 5.1. The various components of the total search latency are studied in Section 5.2, followed by measures to improve LDAP performance. Some important limitations on LDAP performance are studied in Section 5.3. We then discuss the effect of session re-use on server performance in Section 5.4. Finally, performance of update operations is compared to the performance of search operations in Section 5.5.

5.1 Overall LDAP Performance

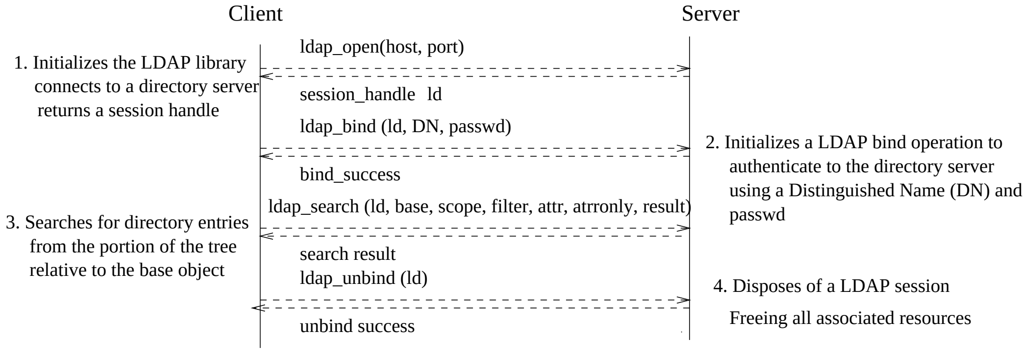

Figure 5: Sequence of steps in a simple LDAP search operation

The general variation in the LDAP latency and throughput as a function of server load are shown in Fig. 6. The load was generated by 1000 clients querying periodically, with each client request retrieving a 488 byte SLAPolicyRules entry from a database with 10,000 entries. Fig. 6 (a) shows the variation in connect, processing, and response latencies as a function of the average request rate at the LDAP server, and Fig. 6 (b) shows the variation in throughput, or number of queries served per second, as a function of the average request rate. Below a load threshold corresponding to a request rate of 105 per second, response latency remains fairly constant at approximately 64 ms, and is dominated by the processing time of 60 ms. Above this threshold, the response time increases rapidly with increasing load. In this region, the connect time is seen to be the main bottleneck; the processing time also increases with load, but its effect is less significant. Corresponding to the latency characteristics, Fig. 6 (b) shows that the server throughput saturates at a rate of approximately 105 requests per second. We now consider various aspects of this performance in more detail.

5.2 Improving the LDAP Search Performance

In this section, we first investigate the various components of the search and connect latency. We then consider two important measures to improve effective search latency and throughput: disabling the Nagle algorithm [16] implemented in TCP, and the caching of LDAP entries.

5.2.1 Components of LDAP Search Latency

We now consider the components of the search latency in more detail. These results were obtained by adding monitoring code to the various process modules in the slapd daemon. Fig. 7 shows the major components that contribute to the server and client search latency, under a load of 105 search requests/second, at which the server is not yet saturated. Surprisingly, while the processing time as measured at the client is approximately 60 ms, results obtained from *tcpdump* and from the monitoring code in *slapd* show that out of 60 ms, approximately 50 ms is a waiting time arising from the Nagle algorithm implemented in TCP. We discuss this in greater detail in Section 5.2.3, and consider the components of the actual search latency in this section.

At the server, the LDBM back-end uses an index mechanism to store and retrieve information. Each entry is assigned a unique ID, used to refer to the entry in the indexes. A search for entries first returns a list of IDs of entries that have the value being searched; the IDs are then used to retrieve the corresponding entries. The candidate ID lookup and the data entry retrieval are seen to take up around 5 ms, 60% of the total search latency of 8.2 ms.

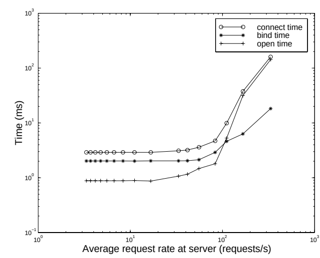

Figure 8: Variation of the *open* time and *bind* time components of the average connection time with average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes.

The main front-end operations at the server are building the search filter, testing a search filter against an entry, ASN.1 encoding of the result entry, sending the search result, and sending the search status. The client-side LDAP library processing includes building the search request, ASN.1 decoding, and construction of the final search result. In all, the front-end operations take around 3.16 ms, 40% of the total response latency. 36.8% of the front-end latency is contributed by ASN.1 data encoding, followed by sending status information, 7.5%, building the search request, 7%, ASN.1 decoding, 6.7%, testing a search filter against an entry, 6.2%, forming the search filter, 5.7%, and sending the search result, 5.4%. The remaining operations, including the ASN.1 encoding and decoding of the query and other information, occupy the remaining 25% of the front-end latency. Profiling of the slapd daemon also showed that at heavy loads, the increase in the total response time is due to the CPU contention among competing threads.

5.2.2 Components of LDAP Connect Latency

The connect latency has two components, corresponding to the *ldap_open* and *ldap_bind* steps of the LDAP search operation shown in Fig. 5. *ldap_open* initializes the LDAP library, opens a connection to the directory server, and returns a session handle for future use. Thus, the open time mainly consists of the session set up time and the TCP connection time, as shown in Fig. 7.

In version 2 of the LDAP protocol, the *ldap_bind* step (client authentication) is a mandatory requirement. As mentioned previously, LDAP supports different authentication methods, and simple password-based authentication was used in the experiments reported here. Fig. 7 shows that the server takes 80% of the binding time, and the client takes 20% of the time.

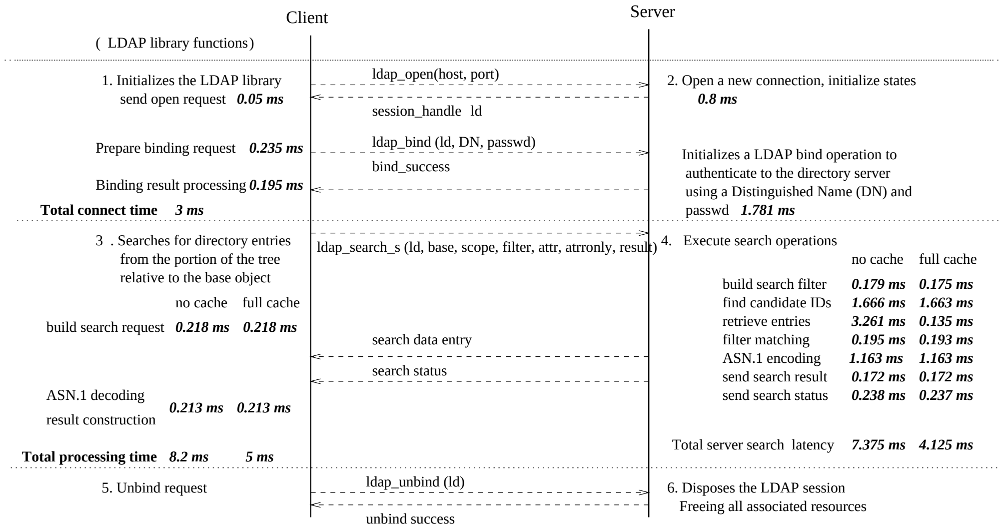

Figure 7: Latency associated with the various server and client process modules in a LDAP *search* operation

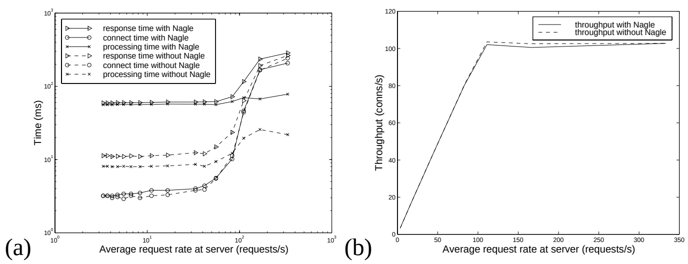

Figure 9: Comparison of the server performance with and without Nagle algorithm: (a) average connection time, processing time and response time; (b) average server throughput. The directory had 10,000 entries and each entry size was 488 bytes.

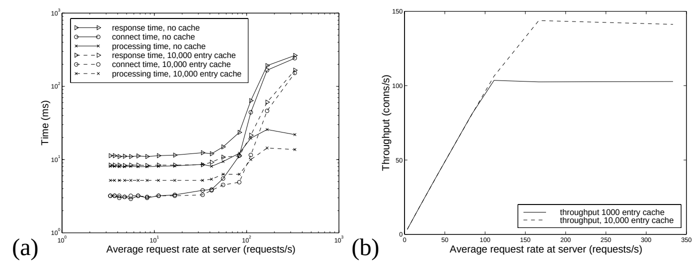

Figure 10: Comparison of the performance when the server is configured with 0 and 10,000 entry caches: (a) the response latency; (b) the throughput.

Fig. 8 shows the open time and authentication time components of the connect time, as a function of the request rate. At small loads, the authentication time is more than twice the open time and dominate the connect latency, which is consistent with the profiling results. The increase in connect time beyond a request rate of 105 per second is largely dominated by the increase in open time.

5.2.3 Effect of the Nagle Algorithm on Search Latency

The Nagle algorithm restricts sending of packets when the segment available to send is less than a full MTU size, in order to reduce transmission of small packets and thus improve network utilization. The algorithm works as follows: if all outstanding data has been acknowledged, any segment is sent immediately. If there is unacknowledged data, the segment is only transmitted if it is a full MTU size. Otherwise, it is queued in the hope that more data will soon be delivered to the TCP layer and then a full MTU can be sent. Fig. 9 shows that when the Nagle mechanism is disabled by enabling the TCP_NODELAY socket option, the LDAP search time is reduced from 60 ms to around 8 ms, while the throughput remains unchanged.

slapd responds to a search request from the client side ldap library functions in two steps: it first returns the data entries; it then sends the search and transmission status. The client side ldap library functions then construct the final results and send to the LDAP client. The results from *tcpdump* indicated that the sending of search status information (14 byte) was delayed about 50 ms until an acknowledgement message was received from client side.

Researchers have presented evidence [17, 18] that the Nagle algorithm should be disabled in order to reduce latency and to protect against unforseen interactions between TCP and HTTP with persistent connections. We believe Nagle algorithm should also be turned off in the LDAP application, since the delay of the last segment of the response was shown to be unnecessary from the *tcpdump* results. To avoid consequent extra packets on the network, functions such as writev() can be used as have been used in WWW servers such as Flash and JAWS.

5.2.4 Effect of Caching LDAP Entries

The LDBM backend can be configured to keep a cache for each index file (*dbcachesize*) and a cache of entries (*cachesize*) in memory. We compared the server performance with two different sizes of the index cache, 100 KB and 10 MB. The backend search time was almost identical for either case, although at the smaller cache size there was a large increase in the time required to populate the directory. This indicated that the index cache was large enough to hold the index files in either case.

We studied the impact of the entry cache size on the server performance by varying the cache size from 0 entries (out-of-memory directory without any cache) to 10,000 entries (in-memory directory with full cache) while keeping the index cache size at 10 MB. For a directory with 10,000 entries, a cache with 10,000 entries results in a reduction in the back-end entry retrieval latency (shown in Fig. 7) from 5 ms to 1.8 ms, with little change in the other latency components. Consequently, the total processing time reduces from 8.2 ms to 5 ms, and the contribution of the back-end processing time to the total latency reduces from 60% to 36%. When the latency is plotted as a function of load, we see a 40% reduction in search processing time due to caching over nearly the entire range of loads, as shown in Fig. 10 (a). Correspondingly, the throughput increases 25% and reaches 140 requests/second, as shown in Fig. 10 (b). The relatively smaller improvement in throughput compared with the improvement in processing time is because at high loads, the connect latency is the dominant component in the total response latency. Memory usage was observed to increase 9% with the above increase in cache size.

Since we are primarily interested in the performance of the LDAP server, in the experiments in the remainder of this paper, the Nagle algorithm was disabled to eliminate the unnecessary wait before returning search results to the client. Unless otherwise specified, an LDBM index cache of size 10 MB, and entry cache equal to the size of the directory were used. Before an experiment, the cache was first filled up to avoid the extra overhead due to cache misses at the beginning of the experiment.

5.3 Performance Limitations

In this section, we study the limitations of the LDAP server performance in three important areas: server CPU capability, the scaling of the LDAP directory, and the scaling of LDAP entry size. An useful point to consider is that in some cases network connectivity may significantly influence the perceived server response. In our case, since clients were connected to the server over a high-speed LAN, this effect could be neglected.

5.3.1 Server Processor Capability: Single vs Dual Processors

In this section, we consider the importance of processing capability in determining the server performance. As mentioned earlier, all our experiments were carried out on a dual processor server, but by binding the *slapd* process to one of the two CPU's, it was used in single-processor mode for the experiments in other sections. To determine the influence of processor capability, we performed some experiments to compare the performance in single-processor mode with the performance in dual-processor mode.

Fig. 11 (a) shows the comparison of latency versus connection rate characteristics for the two cases, using a load generated by search operations. The dual processor server shows similar performance at low loads, and the advantage increases to give roughly 40% smaller latency at higher loads for the total response time. The reduction in latency is observed mainly due to the reduction in connect time. The processing time due to search actually increases slightly at heaviest load, which may be due to the memory contention between the two processors.

Fig. 11 (b) shows the comparison of throughput characteristics for single and dual processors. As seen earlier, the throughput of the single processor server starts to saturate beyond a load of 105 requests per second, and saturates at 140 requests per second. The dual processor server starts to saturate beyond a load of 155 requests per second, and does not saturate completely even at a load of 194

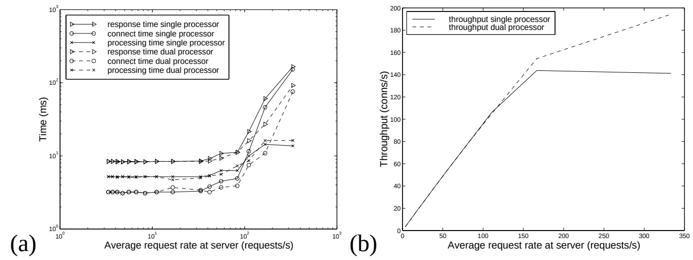

Figure 11: Effect of using a dual processor server on the response time and throughput, with a directory containing 10,000 entries, entry size 488 bytes: (a) connect, processing and response latencies versus request rate; (b) throughput versus request rate

requests per second. Also, at the highest load, CPU utilization in the single-processor server reached 98%, while CPU utilization in the dual processor remained less than 50%.

The above results suggest that the improvement given by the dual processor server would continue to increase at higher loads than those used in the experiment. Higher loads were obtained in an experiment with each client generating requests continuously (a new query upon completion of the previous query) instead of at periodic intervals. Read and write throughput characteristics were generated using LDAP *search* and *add* operations respectively. The results are shown in Fig. 12. Consistent with trends observed in Fig. 11 (a), at low loads the throughput characteristics are similar for dual processor and single processor servers. Beyond a threshold load of about 8-10 clients, the write throughput saturates at around 60 connections per second for the single processor server and 85 connections per second for dual processor operation, an improvement of roughly 40%. A similar effect is observed in the read throughput, which reaches saturation with just 4 clients and gives 150 connections per second for single processor server and 211 connections per second for dual processor server, an improvement of roughly 40%, the same improvement rate as the write operation.

There is a second load threshold in write throughput of about 36 clients for single processor and 48 clients for dual processors beyond which the throughput decrease with increase in load, while the read throughput remains constant within the load range of the experiments. The reduction in throughput of write operations may be due to the increasing contention of system resources among children processes and the increase in the network delay when the server loads increase. These experiments also show the throughput of *search* operations is roughly 2.5 times that of *add* operations in both the single processor and dual processor case.

Overall, processor capability plays a major role in limiting system performance for an in-memory directory, and using a dual processor server gives a significant performance benefit.

5.3.2 Scaling of Directory Size

In this section, we study the scaling of LDAP performance with the directory size, and discuss the limitations on performance at large directory sizes.

We first consider the scaling up of the directory when the directory is in-memory. Fig. 13 (a) shows the comparison of response latency of a directory with 50,000 entries and 50,000 entry cache, with a

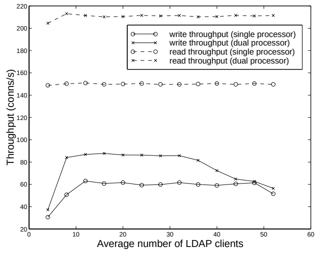

Figure 12: Throughput of search and add operations for single and dual processor servers, with a varying number of LDAP clients generating queries continuously

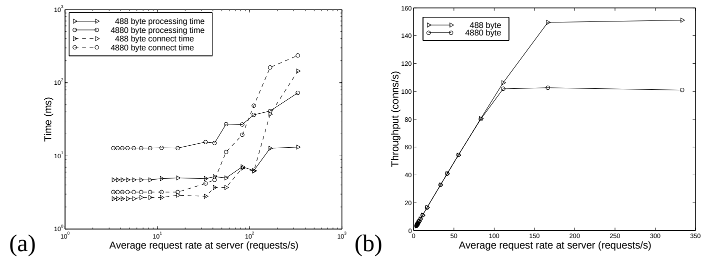

Figure 14: Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for 5000 entry directory and 5000 entry cache

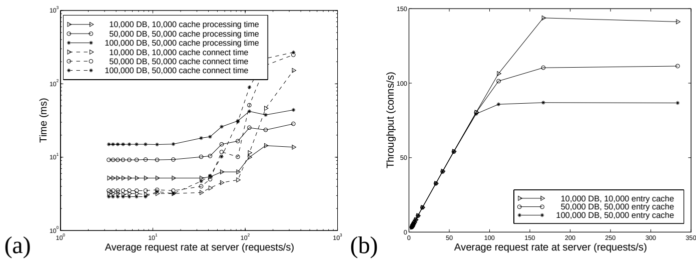

Figure 13: Effect of directory size on the server connect and processing times (a) and the server throughput (b).

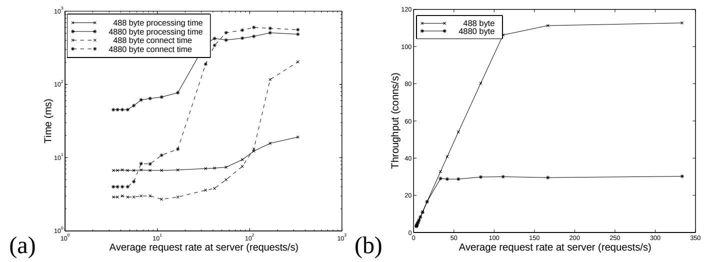

Figure 15: Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for a 10,000 entry directory, 5000 entry cache

directory with 10,000 entries and 10,000 entry cache. The increase in the total response with directory size is mainly due to the increase in the processing time, which increases by 60%, from 5 ms to 8 ms. Profiling at the server shows that the increase in the processing time is, as expected, due to increase in back-end processing. Specifically, it is mainly due to the increase in the entry retrieval time (by 2.6 ms), and a slight increase in the ID search time (by 0.2 ms). Fig. 13 (b) shows that the throughput decreases 21% and saturates at 110 requests/second with the increase in directory size.

As the directory size increases, the database cache size is eventually limited by the available system RAM, and the directory can no longer be kept in-memory. In our system, when the cache size was increased beyond 50,000, performance degraded progressively due to lack of memory. When the directory is out-of-memory, performance scaling is limited both by the database search time, and by the disk access time. Fig. 13 shows that further increasing the number of directory entries from 50,000 to 100,000 while keeping the cache size at 50,000 entries causes the response time to increase another 7 ms (87.5%) and the total processing time reaches 15 ms. Correspondingly, the throughput reduces from 110 requests/second to 85 requests/second (23%). Since the connect latency dominates the response at high loads, the reduction in throughput is relatively small compared to the increase in processing time.

To summarize, there was a moderate deacrease in latency and throughput with directory scaling up to 50,000 entries, due to an increase in database search time. Further scaling was constrained by system memory leading to an out-of-memory directory, and the deterioration in processing latency was significantly sharper, due to increasing disk access time and database search time.

5.3.3 Scaling of the Directory Entry Size

In this section we study the scaling of performance with the size of the LDAP entry. In our experiments, we compared the performance for LDAP directories with 488 byte entries, and with 4880 byte entries. For the larger entry size, the availability of system RAM limited the maximum database cache size to 5000 entries, beyond which the performance degraded rapidly. We first performed the comparison for an in-memory directory, using a 5000 entry directory and 5000 entry cache for both entry sizes. Fig. 14 (a) shows the processing latency, and Fig. 14 (b) the throughput, as a function of load. At light and moderate loads, the total response latency increases by about 8 ms with increase in entry size, while the throughput remains the same. Profiling at the server shows that the increase in response time comes mainly from the increase in the ASN.1 encoding time, from 1.2 ms to 7 ms. In addition, the filter matching step took around 0.8 ms at the higher entry size, up from 0.2 ms at the lower entry size. These results are understandable, since both ANS.1 encoding and the filter matching process depend on the number of attributes in the LDAP entry, which increases as the entry size increases. The latency of the other front-end processing steps, and back-end processing steps increase by much smaller amounts. Under heavy loads, when the server saturates, Fig. 14 (b) shows the maximum throughput at 4880 bytes is 30% smaller.

The comparison between the two entry sizes was also performed with a directory of 10,000 entries, the cache size remaining at 5,000 entries. Fig. 15 (a) shows the processing latency, and Fig. 15 (b) the throughput, as a function of load. The increase in total response time with increase in entry size is now 40 ms at low and moderate loads. The much larger detrioration in response time with entry size is due to the increased data transfer time from the disk in addition to the the increased front-end processing time. The increase in front-end processing time remains at about 8 ms, confirming that the front-end processing is not influenced greatly by the increase in directory size, and the increase in data handling time at the back-end is 32 ms. Also, for the larger entry size, the processing time is comparable with the connect latency even at high loads (unlike previous

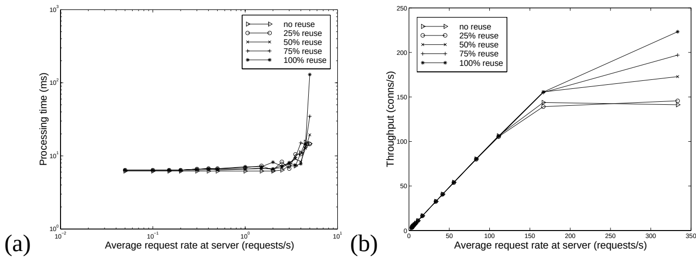

Figure 16: Effect of session reuse rate on the server processing times (a) and the server throughput (b).

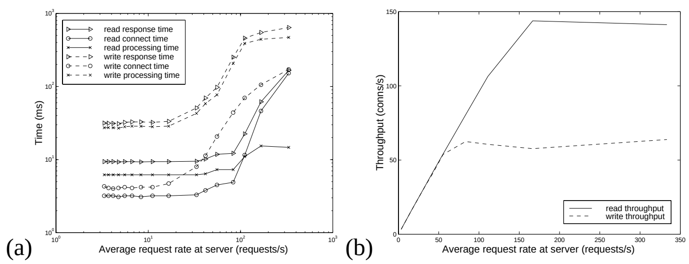

Figure 17: Comparison of latency and throughput characteristics for search and add operations: variation of connect, processing and response latency with server load (a), and variation of server throughput with server load (b). (Directory had 10,000 entries, entry size 488 bytes)

experiments in which the connect latency dominated the total response at high loads). Consequently the increase in processing time with entry size strongly influences the maximum throughput, which decreases sharply from 110 requests/second to 30 requests/second.

To summarize, for an in-memory directory the scaling of the total processing latency with entry-size is determined by the front-end processing, specifically, by the ASN.1 encoding and filter matching. For out-of-memory operation, the increase in processing latency is dominated by the increased back-end data retrieval time. The maximum throughput also decreases sharply in this case.

5.4 Session Reuse

In the experiments so far, the client requested a single search or add operation, and then closed its connection. We now study the change in performance when a client performs multiple searches in a LDAP session. We varied the degree of session reuse from 0 to 100%, with 0% corresponding to a new connection being established for every search, and 100% corresponding to the connection being left open during the duration of the experiment.

Fig. 16 (a) shows that the search time increases at heavy loads as the session reuse rate increases from 0% to 100%. This is because the actual server load for searching operations increases with the increase in session reuse, while the total connect latency decreases because fewer connect requests are received. At the same time, the total response time decreases as can be deduced from the increase in the throughput shown in Fig. 16 (b) from 105 request/second to 155 requests/second at the onset of congestion., and from 140 requests/second to 223 requests/second under the heaviest load, an improvement of 60%.

5.5 Performance under Update Load

The experiments described so far measured the performance under loads generated by *search* requests. In order to compare these results with the performance under update operation, a set of experiments was performed with clients each periodically querying the server with *add* requests. Fig. 17 shows that the processing time for *add* operations dominates the total response time not only at low loads, but also when the server saturates. This is unlike the search scenario, in which the connect time dominates the total response time when the server is under heavy load. This is probably because the latency due to locks generated by competing *write()* requests becomes significant under heavy load. At low loads, the *add* latency is about four times the search latency, and the difference between the two decreases at high loads, as the connect latency becomes more important. Fig. 17 (b) shows that the add throughput begins to saturate beyond a threshold of 60 requests per second and finally saturates at 65 requests per second, about 55% less than the search throughput.

The LDAP *modify* operation was not directly investigated. A *modify* operation involves searching and retrieving an existing entry, modifying it, and then writing it back to the LDAP directory. Consequently, one would expect the processing latency in this case to be roughly the sum of the *search* and *add* processing latencies.

6. RELATED WORK

Benchmarking of LDAP server performance has been reported recently by Mindcraft [19]. The purpose of Mindcraft's work is to compare the performance of three servers: Netscape Directory server 3.0 (NSDS3), Netscape Directory Server 1.0 (NSDS1) and Novell LDAP Services (NDS). In their work, the performance for a 10,000 entry personnel directory was measured on a 200 MHz Intel Pentium Pro with 512 MB RAM. The throughput for NSDS3, NSDS1, and NDS were found to be 183 requests/second, 38.4 requests/second and 0.8 requests/second respectively. The size of the LDAP entry was not specified. The directory was in-memory in all cases, and the performance with larger, out-of-memory directories (or large entry sizes) was not considered. Since the directory was in-memory, CPU capability was found to be the bottleneck.

In the above work, the LDAP server was generally treated as a black box. Our work differs significantly in our objectives and approach. We have determined the scalability of the performance particularly with respect to directory size and entry size, determined the contribution of different system components and parameters to the server performance and scalability, and provided suggestions for improving system performance. The detailed nature of this work also dictated the choice of OpenLDAP instead of a commercial server, as explained earlier.

7. CONCLUSIONS

In this paper, we have discussed the performance of the LDAP directory service in a dynamic, distributed environment, with frequent directory accesses. The LDAP directory structure used in the experiments is based on a proposed LDAP schema for administration of Service Level Specifications in differentiated service networks, and a brief explanation of the use of LDAP in this context has been provided. However, the experimental results are applicable to LDAP directory applications in general.

We have shown that under our normal operating conditions - a directory with 10,000 488 byte entries, and a cache size of 10,000 entries - the LDAP server has a response latency of 8 ms at loads up to 105 search requests per second, and a maximum throughput of 140 search requests per second. Out of the total response latency of 8 ms, 5 ms comes from the processing latency, 36% of which is contributed by back-end processing (entry retrieval from the database), and 64% by front-end processing. In general, at high loads, the connect latency increases sharply to dominate the overall response, and eventually limits the server throughput. Consequently, a change in the processing time due to changes in system parameters has a relatively smaller effect on the maximum throughput.

In addition to this basic performance specification, we have obtained a detailed profile of contributions of various system components to the overall performance; studied the scaling of performance with directory size, entry size, and session re-use; and determined the relative importance of various factors in determining scalability, namely front-end versus back-end processing, CPU capability, and available memory. We have also identified an important required modification to the basic OpenLDAP implementation in order to obtain the above performance. We now briefly summarize our important findings.

- *Disabling of Nagle algorithm*. The Nagle algorithm was observed to contribute an additional wait time of roughly 50 ms to a search operation. Disabling the Nagle algorithm results in a reduction in response time by a factor of 7 under normal operation.

- *Entry caching*. For a directory with 10,000 entries, an in-memory directory (cache-size 10,000) has a 40% improvement in processing time and 25% improvement in throughput over a directory without a cache.

- *Scaling with directory size*. The scaling of performance with the number of entries in the directory is determined by the back-end processing. Up to 50,000 directory entries can be kept in-memory in our system, and the server processing time and throughput deteriorate by about 60% and 21% respectively when directory size increases from 10,000 to 50,000. Beyond this limit, the directory is out-of-memory due to system RAM constraints, and increasing the directory size from 50,000 to 100,000 entries results in a sharper increase in processing time of another 87.5%, and a decrease in throughput by 23%.

- *Scaling with entry size*. The scaling of performance with the entry size in the directory is determined by the front-end processing, mainly an increase in the time for ASN.1 encoding of the retrieved entry, as long as the directory is in-memory. An increase in entry size from 488 bytes to 4880 bytes for a 5,000 entry directory results in an increase in processing time of 8 ms at moderate load, 88% of which is due to the increased ASN.1 encoding time, and a throughput deterioration of about 30%. However, for a directory with 10,000 entries, the cache size is still limited to 5,000 by the system RAM, and a similar increase in entry size results in a much larger throughput deterioration of about 70%, mainly due to the increase in data transfer latency from the disk.

- *CPU capability.* For an in-memory directory, the CPU is a significant bottleneck. Using dual processors improves performance by 40%.

- *Session re-use.* In general, higher session re-use leads to improved performance. A 60% gain in performance is obtained when the session is left open during the duration of the experiment, relative to when a connection is opened and closed for each search request.

In conclusion, we believe that the results show that OpenLDAP slapd is a potential candidate for supporting policy administration in the differentiated service environment as well as in other network applications that need dynamic directory access support. In future, we plan to evaluate other LDAP servers based on the criteria developed in this paper, which may run on different platforms and with different implementations.

8. REFERENCES

[1] A. Dun, D. Hennessy, and F. Dawson Jr., “Calendar attributes for vcard and LDAP,” Internet Draft, Internet Engineering Task Force, Mar. 1998. Work in progress.

[2] B. Aboba, “Lightweight directory access protocol (v3): Schema for the routing policy specification language (RPSL),” Internet Draft, Internet Engineering Task Force, Nov. 1997. Work in progress.

[3] B. Aboba, “Extension for PPP authentication,” Internet Draft, Internet Engineering Task Force, Nov. 1997. Work in progress.

[4] L. Bartz, “LDAP schema for role based access control,” Internet Draft, Internet Engineering Task Force, Oct. 1997. Work in progress.

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSetVation Protocol (RSVP) Version 1 Functional Specification,” RFC2205, Sept. 1997.

[6] K. Nichols and S. Blake, “Definition of the Differentiated Services Field (DS Byte) in the IPv4 and IPv6 Headers”, Internet Draft, May 1998.

[7] S. Blacke, et al, “A Framework for Differentiated Services,” Internet-Draft, Internet Engineering Task Force, Feb., 1999. Work in progress.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss,“An Architecture for Differentiated Services”, RFC 2475, Dec. 1998

[9] W. Yeong, T. Howes and S. Kille, “Lightweight Directory Access Protocol,” RFC1777, Mar. 1995.

[10] M. Wahl, T. and S. Kille, “Lightweight Directory Access Protocol V3,” RFC 2251, Dec. 1997.

[11] The Open Source LDAP suite, http://www.OpenLDAP.org

[12] The Berkeley Database, http://www.sleepycat.com

[13] E. Ellesson, et. al. “Schema for Service Level Administration of Differentiated Services and Integrated Services in Networks,” Internet-Draft, Internet Engineering Task Force Internet-Draft, June 1998.

[14] R. Rajan, J. C. Martin, S. Kamat, M. See, R. Chaudhury, D. Verma, G. Powers, R. Yavatkar, “Schema for Differentiated Services and Integrated Services in Networks,” Internet Draft, Internet Engineering Task Force, Oct. 1998. Work in progress.

[15] M. Beigi, R. Jennings, S. Rao, D. Verma, “Supporting Service Level Agreements using Differentiated Services,” Internet Draft, Internet Engineering Task Force, Nov. 1998. Work in progress.

[16] John Nagle, “Congestion control in IP/TCP internetworks,” *ACM Computer Communication Review*, vol. 14, no. 4, pp. 11-17, Oct. 1984.

[17] J. Heidemann, “ Performance interactions between P-HTTP and TCP Implementations,” *ACM Computer Communication Review*, vol. 27, no. 2, pp. 65-73, April 1997.

[18] H. F. Nielsen, et. “Network Performance Effects of HTTP/1.1, CSS1, and PNG,” *ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Cannes, France, Sep. 1997.

[19] Directory Server Certified Performance Reports, http://www.mindcraft.com/perfreports/ldap/.