1990

Directory Location Service (DLS)

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Report Number:
90-1010

DIRECTORY LOCATION SERVICE (DLS)

Douglas E. Comer

CSD-TR 1010
September 1990

Directory Location Service (DLS)

Douglas E. Comer
Computer Science Department
Purdue University
West Lafayette, IN 47907

CSD-TR 1010
September 1990

This work was supported in part by a grant from the National Science Foundation (ASC-851-5851), with additional support from Purdue University.

Directory Location Service:
Overview and Access Protocol Specification

Directory Location Service

The Directory Location Service (DLS) provides information on the location (i.e., addresses, and protocols needed to access) white pages name servers. The intention of DLS is to provide a top-level service that helps users limit the scope of a white-pages query to only pertinent servers. Conceptually, DLS is a centralized database that contains one entry per organization; each entry describes the nameservers that the organization operates. A user contacts DLS and presents it the name of an organization to find a set of servers pertinent to a given query.

The DLS design represents a compromise between two extremes: (1) exhaustive search of all possible white pages servers and (2) limited search of a small fixed set of servers. The former is inadequate because the search takes arbitrarily long; the latter because specifying a small set of servers a priori makes it impossible to look up arbitrary names. DLS is late binding in the sense that it does not require the user to prebind a list of nameservers to be searched. Instead, a user presents keyword(s) to DLS that describe the organization associated with a particular name. To eliminate spelling errors, DLS matches the keyword(s) with the keys on entries in its database using a phonetic match algorithm. The user interface receives information on all records that match, and either contacts the servers specified in them or asks the user to select a subset that narrows the search further.

Architecture

The DLS architecture consists of three conceptual pieces: a user interface, local server, and root server. A user interacts with the user interface, which may use simple text or complex graphics. The user interface includes client software that contacts the local server at the site to request information. The local server accepts requests from client(s), contacts the root server to find the information, and returns results to the appropriate client. Most important, local servers cache results and return answers from the cache when possible.

Stored Data

Information stored at the server is classified as one of two record types: key information records or data records. Key information records describe the syntax of keys (e.g., a key information record might tell a user that types "IBM" to rephrase the query as "IBM-site", where the site can be one of "yorktown", "san jose", etc.). Each subscriber organization† has one data record that contains a list of the directory servers the organization supports. Each server is identified by a name, address, access protocol, and transport protocol. For example, an entry might say that the server is an Internet Domain Name Server named "pendragon.cs.purdue.edu" with IP address 128.10.2.5, access protocol "DNS", transport protocols (and ports) "UDP/53" and "TCP/53".

Conceptual Sequence of Matched Items

We think of all the DLS database entries that match a given request as forming a sequence. Conceptually, items in the sequence are numbered starting at *1*, making it possible to identify items by giving their integer position.‡ If the client uses TCP to connect to the server, the server returns all entries that match a query. However, if the client uses UDP to contact the

---

†We do not define what an "organization" is except to say that it should have a single name or well-known identifying phrase, be large enough to operate a white pages server, and have a single owner or administrative authority. Thus, a university might be an organization.

‡Note: because the database will not change rapidly, we assume that we do not need to worry about the sequence changing size during a client-server interaction.

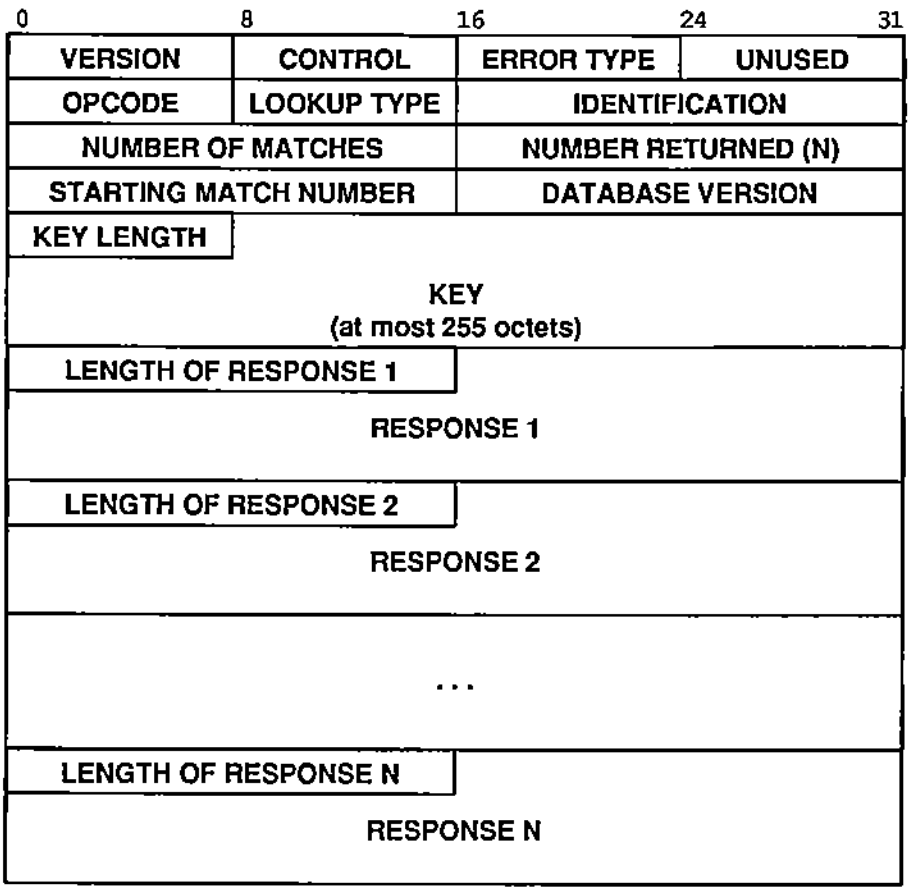
server, the server cannot return the complete sequence in one datagram. Instead, it sends an initial response that describes the size of the sequence and sends however many entries fit in the datagram. The client must then contact the server repeatedly to ask for additional entries.

An exception is made for requests that do not match any data entries. If the server finds information entries for the key, it will return them, even though the client asked for data records.

It is also possible to request the keys themselves instead of data records attached to the keys. In the simplest case, we think of a conceptual sequence consisting of all key values (i.e., the set of all primary and secondary keys sorted into lexicographic order with all duplicates removed.) Individual requests can specify subsets of the keys (e.g., the subsequence of keys starting at the letter "f").

Message Format

All messages exchanged between a DLS client and a DLS server have a format as illustrated in the following diagram.



Fields of a DLS Message

•**VERSION.** Specifies the version of the DLS protocol; the version described in this document is *5*. The first octet of all successive versions of the protocol will contain a version number.

•**CONTROL.** Contains 8 bits that control processing by the server. The meaning of each bit is defined in Table 1.

CONTROL Bit	Meaning If Set
0	**Error.** A client must never set this bit. A server sets this bit to report an error. The *ERROR TYPE* field (see below) contains further information about the error.
1	Reserved.
2	**Request/Reply.** Client clears this bit in a request and the server sets this bit in a reply. A client that receives a message with this bit cleared or a server that receives a message with this bit set generates an error with *ERROR TYPE* set to *2*.
3	**Count Only.** Client sets this bit to request a response that the server return only a count of records in the response but no actual data. Server leaves this bit set in the reply and returns the message header, including the number of matches, but no actual data (i.e., the *NUMBER RETURNED* field will be zero).
4	**Authoritative Answer.** Client sets this bit to request authoritative answers only; server sets this bit whenever it sends an authoritative answer (i.e., an answer that did not come from the cache).
5	**No Delay.** Client sets this bit to specify that no answer is required if the server must delay. Server sets *error* control with *ERROR TYPE* set to *5* to reply if it cannot provide an answer without delay.
6	**Inform Delay.** Client sets this bit to request that the server inform it if the server expects a delay in answering a request. Upon receipt of a message with this bit set, a local server sends an immediate reply with this bit set. The reply contains zero or more responses that are available immediately (i.e., from the cache). The server sends additional responses later when they become available (e.g., when the local server receives a reply from the root server).
7	**Version Check.** This bit is used only by clients that maintain a cache). The client sets this bit and fills in the *DATABASE VERSION* field to specify that the lookup should only occur if the client's version number is out of date with respect to the server's version number. If the client's version number is up to date, the server sets this bit in the reply and does not send data. For opcode *3* (see Table 3 below), the version refers to the root version number; for all other queries it refers to the local server version number.

Table 1. Values for CONTROL bits (numbered left-to-right) and their meanings.

•**ERROR TYPE.** Contains a code that specifies the type of error according to Table 2. All error messages carry a nonzero *ERROR TYPE* and are 16-octets long. Thus, an error message includes fields in the header through the *DATABASE VERSION*. Clients use the contents of these fields to associate the error message with the query that caused it.

ERROR TYPE	Meaning
0	**No Error.** The value clients send in queries and the value servers send if no error has occurred.
1	**Version Mismatch.** The version number on a request does not match the version of the server software that received it. This value is guaranteed for all versions of the protocol. (The reason for making this guarantee is so DLS client software can always understand error messages caused by a version mismatch, even if the client cannot understand the remainder of the message).
2	**Header Error.** One or more field(s) in the query were incorrect (e.g., contained a value out of bounds).
3	**Unauthorized Query.** The client did not possess (or supply) correct authorization for the query.
4	**Service Not Supported.** The server to which the query was sent did not supply the requested service.
5	**Delay Needed.** The server could not respond to a query immediately but the client specified no delay in the *CONTROL* field (see above).
6	**Sequence Out Of Range.** The value the client specified in field *STARTING MATCH NUMBER* was not in the range of the sequence of matches.
7	**Other error.** Used for errors other than those above.

Table 2. Possible values for the ERROR CODE field and their meanings.

•**UNUSED.** This field is currently unused and must be assigned zero in both queries and replies.

•**OPCODE.** Specifies an operation as defined in Table 3.

Opcode	Meaning
1	**Request By Key.** Client supplies a key to be used in the search.
2	**Request By Unique ID.** Client is asking for a match using the record's unique ID instead of a search key. This opcode can only be used with lookup types *1* and 2.
3	**Request All.** Client requests all data from database. *KEY* field contains an authorization code (i.e., password). If authorization is correct data records sent will contain ALL fields.

Table 3. Values for the OPCODE field and their meanings.

•**LOOKUP TYPE.** Specifies the type of the request desired or the type of the reply sent, with values assigned according to Table 4.

Type	Meaning
1	**Full Data.** The client requests the server to send all data records that match the query. The server includes full data records in its reply.
2	**Abbreviated Data.** The client requests the server to supply abbreviated data records containing only the unique id, organization name & address, and keywords. The server responds with abbreviated records.
3	**Keyword Information.** The client requests the server to return keyword information records that specify correct (canonical) synonyms for keywords, or notes on how to form keywords. For example, the client might request information about key "IBM" and learn that the general form is IBM-*site*. The server returns keyword information.
4	**Key Sequence.** Client requests items from the sequence of all possible keywords (in lexicographic order) instead of the sequence of records. If *KEY LENGTH* is zero, the request refers to the full sequence of all keys. If *KEY LENGTH* is nonzero, the request refers to a subsequence of keys that start at the first key lexicographically greater than or equal to the value in *KEY*. For example, if *KEY* contains "b", the request refers to the sequence of all keys starting with the letters "b" through "z". The idea behind a key sequence request is that it allows local servers or clients to download all possible keywords.

Table 4. Values for the *LOOKUP TYPE* field and their meanings.

•**IDENTIFICATION.** Contains an integer that the client uses to identify a query (or a retransmission); the server returns the identification when it responds to the query.

•**NUMBER OF MATCHES.** Contains an integer that the server uses to report the number of records in the database that match the query.

•**NUMBER RETURNED (N).** Contains an integer that the client uses to limit the number of matches requested. The server uses it to report the number of items returned in a reply. In requests, the value zero is used to specify no limit. For TCP connections, the *NUMBER RETURNED* will be the minimum of the number the client requests and the *NUMBER OF MATCHES*. For UDP, the *NUMBER RETURNED* specifies the number returned in a particular datagram.

•**STARTING MATCH NUMBER.** Contains an integer that the client uses to specify a position in the conceptual sequence of matches where the responses should start. The server uses it to specify the position in the sequence where the responses in this reply begin. As a special case, clients set the *STARTING MATCH NUMBER* to *0* in their initial request. If a match occurs, clients receive in the reply the first *K* items from the sequence of matches. The client then sends a second request with a *STARTING MATCH NUMBER* of *K+1*, and so on. Other values can be used in the initial request; the protocol does not preclude them. However, requesting a value out of range (i.e., requesting responses starting with *N+1* or greater when the sequence contains only *N* items) results in an error reply with *ERROR TYPE 6*. Thus, clients use *0* to avoid receiving an error reply on requests that have no match. Note that there are separate conceptual sequences for: keyword information requests, key sequence requests, and data requests. Also, the sequence numbers that match a request may change if the database version number changes.

•**DATABASE VERSION.** Contains an integer version number that specifies the version of the database from which a response was derived. The database version changes every time data visible to local servers or clients changes. For authorized *Request All* queries, the database version will reflect the true version number used for root server replication.

•**KEY LENGTH.** Contains an integer used to specify the length of the key that follows. Lengths are specified in bytes; keys are not terminated or padded.

•**KEY.** Contains a string that clients send to specify a search. The meaning of the *KEY* field depends on the *OPCODE* and *LOOKUP TYPE* field (see tables above). For example, in queries with *OPCODE 2*, the key contains a unique ID; in those with *OPCODE 3* it contains an authorization.

•**LENGTH OF RESPONSE.** Contains an integer that servers use to specify the number of bytes in a response. A local server that returns information from its cache can send responses with length zero to inform the client that the response that should occur at that position in the sequence is unavailable. The client can request specific missing items again or use the *CONTROL* bits to prevent receiving partial information.

•**RESPONSE.** Contains a response that matches a request. For data requests, the response contains one record in DLS format, including embedded linefeed characters used to separate lines. Responses are not padded or terminated by special characters when placed in a message. Each response for type key sequence lookups contains one keyword from the database. Keys are returned exactly as the subscriber site entered them. However, DLS searches are case insensitive; it is recommeded that any client software that matches keys should use case-insensitive matching as well.

Client and Server Use of Fields

For a typical query, the client fills in the header and sends a *KEY*. It assigns zero to the *DATABASE VERSION* and *NUMBER OF MATCHES*. It does not include any responses. It sets *NUMBER RETURNED* to zero unless it wants to limit the number of responses. Most queries will not have any *CONTROL* bits set.

The server uses the *CONTROL* bits to determine processing. Assuming no special control, it checks the protocol version number, performs the lookup, changes the header, adds zero or more response fields, and returns the result. The server must specify a database version number and fill in the *NUMBER RETURNED* field. It returns the same *IDENTIFICATION* and *KEY* exactly as received.

If the server finds no matches for a given key, it may look up information records using the key and return them. If so, it changes the lookup type to *3* on the reply. In the unlikely event that the client is using UDP and more information records exist for a key than can fit into a single datagram, the client must send subsequent requests for them using lookup type *3* (i.e., the conceptual sequence of data records that match a given key is separate from the conceptual sequence of information records that match the key, and the client must be careful to specify which it wants).