Network Working Group
Request for Comments: 1913
Category: Standards Track

C. Weider
Bunyip
J. Fullton
CNIDR
S. Spero
EIT

February 1996

Architecture of the Whois++ Index Service

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The authors describe an architecture for indexing in distributed databases, and apply this to the WHOIS++ protocol.

1. Purpose:

The WHOIS++ directory service [Deutsch, et al, 1995] is intended to provide a simple, extensible directory service predicated on a template-based information model and a flexible query language. This document describes a general architecture designed for indexing distributed databases, and then applys that architecture to link together many of these WHOIS++ servers into a distributed, searchable wide area directory service.

2. Scope:

This document details a distributed, easily maintained architecture for providing a unified index to a large number of distributed WHOIS++ servers. This architecture can be used with systems other than WHOIS++ to provide a distributed directory service which is also searchable.

3. Motivation and Introduction:

It seems clear that with the vast amount of directory information potentially available on the Internet, it is simply not feasible to build a centralized directory to serve all this information. If we are to distribute the directory service, the easiest (although not necessarily the best) way of building the directory service is to build a hierarchy of directory information collection agents. In this architecture, a directory query is delivered to a certain agent in the tree, and then handed up or down, as appropriate, so that the query is delivered to the agent which holds the information which fills the query. This approach has been tried before, most notably in some implementations of the X.500 standard. However, there are number of major flaws with the approach as it has been taken. This new Index Service is designed to fix these flaws.

3.1. The search problem

One of the primary assumptions made by recent implementations of distributed directory services is that every entry resides in some location in a hierarchical name space. While this arrangement is ideal for reading the entry once one knows its location, it is not as good when one is searching for the location in the namespace of those entries which meet some set of criteria. If the only criteria we know about a desired entry are items which do not appear in the namespace, we are forced to do a global query. Whenever we issue a global query (at the root of the namespace), or a query at the top of a given subtree in the namespace, that query is replicated to "all" subtrees of the starting point. The replication of the query to all subtrees is not necessarily a problem; queries are cheap. However, every server to which the query has been replicated must process that query, even if it has no entries which match the specified criteria. This part of the global query processing is quite expensive. A poorly designed namespace or a thin namespace can cause the vast majority of queries to be replicated globally, but a very broad namespace can cause its own navigation problems. Because of these problems, search has been turned off at high levels of the X.500 namespace.

3.2. The location problem

With global search turned off, one must know in advance how the name space is laid out so that one can guide a query to a proper location. Also, the layout of the namespace then becomes critical to a user's ability to find the desired information. Thus there are endless battles about how to lay out the name space to best serve a given set of users, and enormous headaches whenever it becomes apparent that the current namespace is unsuited to the current usages and must be changed (as recently happened in X.500). Also, assuming one does impose multiple hierarchies on the entries through use of the namespace, the mechanisms to maintain these multiple hierarchies in X.500 do not exist yet, and it is possible to move entries out from under their pointers. Also, there is as yet no agreement on how the X.500 namespace should look even for the White Pages types of information that is currently installed in the X.500 pilot project.

3.3. The Yellow Pages problem

Current implementations of this hierarchical architecture have also been unsuited to solving the Yellow Pages problem; that is, the problem of easily and flexibly building special-purpose directories (say of molecular biologists) and of automatically maintaining these directories once they have been built. In particular, the attributes appropriate to the new directory must be built into the namespace because that is the only way to segregate related entries into a place where they can be found without a global search. Also, there is a classification problem; how does one adequately specify the proper categories so that people other than the creator of the directory can find the correct subtree? Additionally, there is the problem of actually finding the data to put into the subtree; if one must traverse the hierarchy to find the data, we have to look globally for the proper entries.

3.4. Solutions

The problems examined in this section can be addressed by a combination of two new techniques: directory meshes and forward knowledge.

4. Directory meshes and forward knowledge

We'll hold off for a moment on describing the actual architecture used in our solution to these problems and concentrate on a high level description of what solutions are provided by our conceptual approach. To begin with, although every entry in WHOIS++ does indeed have a unique identifier (resides in a specific location in the namespace) the navigational algorithms to reach a specific entry do not necessarily depend on the identifier the entry has been assigned. The Index Service gets around the namespace and hierarchy problems by creating a directory mesh on top of the entries. Each layer of the mesh has a set of 'forward knowledge' which indicates the contents of the various servers at the next lower layer of the mesh. Thus when a query is received by a server in a given layer of the mesh, it can prune the search tree and hand the query off to only those lower level servers which have indicated that they might be able to answer it. Thus search becomes feasible at all levels of the mesh. In the current version of this architecture, we have chosen a certain set of information to hand up the mesh as forward knowledge. This may or may not be exactly the set of information required to construct a truly searchable directory, but the protocol itself doesn't restrict the types of information which can be handed around.

In addition, the protocols designed to maintain the forward knowledge will also work perfectly well to provide replication of servers for redundancy and robustness. In this case, the forward knowledge handed around by the protocols is the entire database of entries held by the replicated server.

Another benefit provided by the mesh of index servers is that since the entry identification scheme has been decoupled from the navigation service, multiple hierarchies can be built and easily maintained on top of the existing data. Also, the user does not need to know in advance where in the mesh the entry is contained.

Also, the Yellow Pages problem now becomes tractable, as the index servers can pick and choose between information proffered by a given server; because we have an architecture that allows for automatic polling of data, special purpose directories become easy to construct and to maintain.

5. Components of the Index Service:

5.1. WHOIS++ servers

The whois++ service is described in [Deutsch, et al, 1995]. As that service specifies only the query language, the information model, and the server responses, whois++ services can be provided by a wide variety of databases and directory services. However, to participate in the Index Service, that underlying database must also be able to generate a 'centroid', or some other type of forward knowledge, for the data it serves.

5.2. Centroids as forward knowledge

The centroid of a server is comprised of a list of the templates and attributes used by that server, and a word list for each attribute. The word list for a given attribute contains one occurrence of every word which appears at least once in that attribute in some record in that server's data, and nothing else.

A word is any token delimited by blank spaces, newlines, or the '@' character, in the value of an attribute.

For example, if a whois++ server contains exactly three records, as follows:

```
Record 1                        Record 2
Template: User                  Template: User
First Name: John                First Name: Joe
Last Name: Smith                Last Name: Smith
Favourite Drink: Labatt Beer    Favourite Drink: Molson Beer
```

```
Record 3
Template: Domain
Domain Name: foo.edu
Contact Name: Mike Foobar
```

the centroid for this server would be

```
Template:         User
First Name:       Joe
                  John
Last Name:        Smith
Favourite Drink:  Beer
                  Labatt
                  Molson

Template:         Domain
Domain Name:      foo.edu
Contact Name:     Mike
                  Foobar
```

It is this information which is handed up the tree to provide forward knowledge. As we mention above, this may not turn out to be the ideal solution for forward knowledge, and we suspect that there may be a number of different sets of forward knowledge used in the Index Service. However, the directory architecture is in a very real sense independent of what types of forward knowledge are handed around, and it is entirely possible to build a unified directory which uses many types of forward knowledge.

5.3. Index servers and Index server Architecture

A whois++ index server collects and collates the centroids (or other forward knowledge) of either a number of whois++ servers or of a number of other index servers. An index server must be able to generate a centroid for the information it contains. In addition, an index server can index any other server it wishes, which allows one base level server (or index server) to participate in many hierarchies in the directory mesh.

5.3.1. Queries to index servers

An index server will take a query in standard whois++ format, search its collections of centroids and other forward information, determine which servers hold records which may fill that query, and then notifies the user’s client of the next servers to contact to submit the query (referral in the X.500 model). An index server can also contain primary data of its own; and thus act a both an index server and a base level server. In this case, the index server’s response to a query may be a mix of records and referral pointers.

5.3.2. Index server distribution model and centroid propogation

The diagram on the next page illustrates how a mesh of index servers might be created for a set of whois++ servers. Although it looks like a hierarchy, the protocols allow (for example) server A to be indexed by both server D and by server H.

```
     whois++               index                   index
     servers               servers                 servers
                           for                     for
                           whois++                 lower-level
                           servers                 index servers
     _______
    |       |
    |   A   |__
    |_______|  \            _______
                \----------|       |
     _______               |   D   |__             ______
    |       |   /----------|_______|  \           |      |
    |   B   |__/                       \----------|      |
    |_______|                                     |  F   |
                                       /----------|______|
                                      /
     _______                _______  /
    |       |              |       |-
    |   C   |--------------|   E   |
    |_______|              |_______|-
                                     \
                                      \
     _______                           \            ______
    |       |                           \----------|      |
    |   G   |--------------------------------------|  H   |
    |_______|                                      |______|
```

Figure 1: Sample layout of the Index Service mesh

In the portion of the index tree shown above, whois++ servers A and B hand their centroids up to index server D, whois++ server C hands its centroid up to index server E, and index servers D and E hand their centroids up to index server F. Servers E and G also hand their centroids up to H.

The number of levels of index servers, and the number of index servers at each level, will depend on the number of whois++ servers deployed, and the response time of individual layers of the server tree. These numbers will have to be determined in the field.

5.3.3. Centroid propogation and changes to centroids

Centroid propogation is initiated by an authenticated POLL command (sec. 5.2). The format of the POLL command allows the poller to request the centroid of any or all templates and attributes held by the polled server. After the polled server has authenticated the poller, it determines which of the requested centroids the poller is allowed to request, and then issues a CENTROID-CHANGES report (sec. 5.3) to transmit the data. When the poller receives the CENTROID-CHANGES report, it can authenticate the pollee to determine whether to add the centroid changes to its data. Additionally, if a given pollee knows what pollers hold centroids from the pollee, it can signal to those pollers the fact that its centroid has changed by issuing a DATA-CHANGED command. The poller can then determine if and when to issue a new POLL request to get the updated information. The DATA-CHANGED command is included in this protocol to allow 'interactive' updating of critical information.

5.3.4. Centroid propogation and mesh traversal

When an index server issues a POLL request, it may indicate to the polled server what relationship it has to the polled. This information can be used to help traverse the directory mesh. Two fields are specified in the current proposal to transmit the relationship information, although it is expected that richer relationship information will be shared in future revisions of this protocol.

One field used for this information is the Hierarchy field, and can take on three values. The first is 'topology', which indicates that the indexing server is at a higher level in the network topology (e.g. indexes the whole regional ISP). The second is 'geographical', which indicates that the polling server covers a geographical area subsuming the pollee. The third is 'administrative', which indicates that the indexing server covers an administrative domain subsuming the pollee.

The second field used for this information is the Description field, which contains the DESCRIBE record of the polling server. This allows users to obtain richer metainformation for the directory mesh, enabling them to expand queries more effectively.

5.3.5. Query handling and passing algorithms

When an index server receives a query, it searches its collection of centroids and determines which servers hold records which may fill that query. As whois++ becomes widely deployed, it is expected that some index servers may specialize in indexing certain whois++

templates or perhaps even certain fields within those templates. If an index server obtains a match with the query "for those template fields and attributes the server indexes", it is to be considered a match for the purpose of forwarding the query.

5.3.5.1. Query referral

Query referral is the process of informing a client which servers to contact next to resolve a query. The syntax for notifying a client is outlined in section 5.5.

5.3.6 Loop control

Since there are no a priori restrictions on which servers may poll which other servers, and since a given server may participate in many sub-meshes, mechanisms must be installed to allow the detection of cycles in the polling relationships. This is accomplished in the current protocol by including a hop-count on polling relationships. Each time a polled server generates forward information, it informs the polling server about its current hopcount, which is the maximum of the hopcounts of all the servers it polls, plus 1. A base level server (one which polls no other servers) will have a hopcount of 0. When a server decides to poll a new server, if its hopcount goes up, then it must information all the other servers which poll it about its new hopcount. A maximum hopcount (8 in the current version) will help the servers detect polling loops.

A second approach to loop detection is to do all the work in the client; which would determine which new referrals have already appeared in the referral list, and then simply iterate the referral process until there are no new servers to ask. An algorithm to accomplish this in WHOIS++ is detailed in [Faltstrom 95].

6. Syntax for operations of the Index Service:

The syntax for each protocol componenet is listed below. In addition, each section contains a listing of which of these attributes is required and optional for each of the componenet. All timestamps must be in the format YYYYMMDDHHMM and in GMT.

6.1. Data changed syntax

The data changed template look like this:

```
# DATA-CHANGED
 Version-number: // version number of index service software, used to
                 // insure compatibility. Current value is 1.0
 Time-of-latest-centroid-change: // time stamp of latest centroid
```

```
                                 // change, GMT
 Time-of-message-generation: // time when this message was generated,
                             // GMT
 Server-handle: // IANA unique identifier for this server
 Host-Name: // Host name of this server (current name)
 Host-Port: // Port number of this server (current port)
 Best-time-to-poll: // For heavily used servers, this will identify
                    // when the server is likely to be lightly
                    // loaded so that response to the poll will be
                    //speedy, GMT
 Authentication-type: // Type of authentication used by server, or NONE
 Authentication-data: // data for authentication
# END // This line must be used to terminate the data changed
                 // message
```

Required/optional table

```
Version-Number  REQUIRED
Time-of-latest-centroid-change  REQUIRED
Time-of-message-generation      REQUIRED
Server-handle   REQUIRED
Host-Name       REQUIRED
Host-Port       REQUIRED
Best-time-to-poll       OPTIONAL
Authentication-type     OPTIONAL
Authentication-data     OPTIONAL
```

6.2. Polling syntax

```
# POLL:
 Version-number: // version number of poller's index software, used to
                 // insure compatibility
 Type-of-poll: // type of forward data requested. CENTROID or QUERY
               // are the only one currently defined
 Poll-scope: // Selects bounds within which data will be returned.
             // See note.
 Start-time: // give me all the centroid changes starting at this
             // time, GMT
 End-time: // ending at this time, GMT
 Template: // a standard whois++ template name, or the keyword ALL,
           // for a full update.
 Field:    // used to limit centroid update information to specific
           // fields, is either a specific field name, a list of field
           // names, or the keyword ALL
 Server-handle: // IANA unique identifier for the polling server.
                // this handle may optionally be cached by the polled
                // server to announce future changes
 Host-Name: // Host name of the polling server.
```

```
 Host-Port: // Port number of the polling server.
 Hierarchy: // This field indicates the relationship which the poller
              // bears to the pollee. Typical values might include
              // 'Topology', 'Geographical", or "Administrative"
 Description: // This field contains the DESCRIBE record of the
                // polling server
 Authentication-type: // Type of authentication used by poller, or NONE
 Authentication-data: // Data for authentication
# END  // This line must by used to terminate the poll message
```

Note: For poll type CENTROID, the allowable values for Poll Scope are FULL and RELATIVE. Support of the FULL value is required, this provides a complete listing of the centroid or other forward information. RELATIVE indicates that these are the relative changes in the centroid since the last report to the polling server.

For poll type QUERY, the allowable values for Poll Scope are a blank line, which indicates that all records are to be returned, or a valid WHOIS++ query, which indicates that just those records which satisfy the query are to be returned. N.B. Security considerations may require additional authentication for successful response to the Blank Line Poll Scope. This value has been included for server replication.

A polling server may wish to index different types of information than the polled server has collected. The POLLED-FOR command will indicate which servers the polled server has contacted.

Required/Optional Table

Version-Number	REQUIRED,	value is 1.0
Type-Of-Poll	REQUIRED,	values CENTROID and QUERY are required
Poll-scope	REQUIRED	If Type-of-poll is CENTROID, FULL is required, RELATIVE is optional If Type-of-poll is QUERY, Blank line is required, and WHOIS++-type queries are required
Start-time	OPTIONAL	
End-Time	OPTIONAL	
Template	REQUIRED	
Field	REQUIRED	
Server-handle	REQUIRED	
Host-Name	REQUIRED	
Host-Port	REQUIRED	
Hierarchy	OPTIONAL	
Description	OPTIONAL	
Authentication-Type:	OPTIONAL	
Authentication-data:	OPTIONAL	

Example of a POLL command:

```
# POLL:
 Version-number: 1.0
 Type-of-poll: CENTROID
 Poll-scope: FULL
 Start-time: 199501281030+0100
 Template: ALL
 Field: ALL
 Server-handle: BUNYIP01
 Host-Name: services.bunyip.com
 Host-Port: 7070
 Hierarchy: Geographical
 # END
```

6.3. Centroid change report

As the centroid change report contains nested multiply-occuring blocks, each multiply occurring block is surrounded *in this paper* by curly braces '{', '}'. These curly braces are NOT part of the syntax, they are for identification purposes only.

The syntax of a Data: item is either a list of words, one word per line, or the keyword:

  ANY

The keyword ANY as the only item of a Data: list means that any value for this field should be treated as a hit by the indexing server.

The field Any-field: needs more explanation than can be given in the body of the syntax description below. It can take two values, True or False. If the value is True, the pollee is indicating that there are fields in this template which are not being exported to the polling server, but wishes to treat as a hit. Thus, when the polling server gets a query which has a term requesting a field not in this list for this template, the polling server will treat that term as a 'hit'. If the value is False, the pollee is indicating that there are no other fields for this template which should be treated as a hit. This field is required because the basic model for the WHOIS++ query syntax requires that the results of each search term be 'and'ed together. This field allows polled servers to export data only for non-sensitive fields, yet still get referrals of queries which contain sensitive terms.

IMPORTANT: The data listed in the centroid must be in the ISO-8859-1 character set in this version of the indexing protocol. Use of any other character set is a violation of the protocol. Note that the base-level server is also specified to use ISO-8859-1 [Deutsch, et al, 1995].

```
# CENTROID-CHANGES
 Version-number: // version number of pollee's index software, used to
                 // insure compatibility
 Start-time: // change list starting time, GMT
 End-time: // change list ending time, GMT
 Server-handle: // IANA unique identifier of the responding server
 Case-sensitive: // states whether data is case sensitive or case
                 // insensitive. values are TRUE or FALSE
 Authentication-type: // Type of authentication used by pollee, or NONE
 Authentication-data: // Data for authentication
 Compression-type: // Type of compression used on the data, or NONE
 Size-of-compressed-data: // size of compressed data if compression
                          // is used
 Operation: // One of 3 keywords: ADD, DELETE, FULL
            // ADD - add these entries to the centroid for this server
            // DELETE - delete these entries from the centroid of this
            // server
            // FULL - the full centroid as of end-time follows
{ // The multiply occurring template block starts here
# BEGIN TEMPLATE
 Template: // a standard whois++ template name
 Any-field: // TRUE or FALSE. See beginning of 6.3 for explanation.
 { // the template contains multiple field blocks
# BEGIN FIELD
 Field: // a field name within that template
 Data: // Either the keyword *ANY*, or
       // the word list itself, one per line, cr/lf terminated,
       // each line starting with a dash character ('-').
# END FIELD
  } // the field ends with END FIELD
# END TEMPLATE
} // the template block ends with END TEMPLATE
# END CENTROID-CHANGES // This line must be used to terminate the
                         // centroid change report
```

For each template, all fields must be listed, or queries will not be referred correctly.

Required/Optional table

```
Version-number  REQUIRED, value is 1.0
Start-time      REQUIRED (even if the centroid type is FULL)
End-time        REQUIRED (even if the centroid type is FULL)
Server-handle   REQUIRED
Case-Sensitive  OPTIONAL
Authentication-Type     OPTIONAL
```

```
Authentication-Data     OPTIONAL
Compression-type        OPTIONAL
Size-of-compressed-data OPTIONAL (even if compression is used)
Operation     OPTIONAL, if used, upport for all three values is required
Tokenization-type       OPTIONAL
#BEGIN TEMPLATE REQUIRED
Template        REQUIRED
Any-field       REQUIRED
#BEGIN FIELD    REQUIRED
Field           REQUIRED
Data            REQUIRED
#END FIELD      REQUIRED
#END TEMPLATE   REQUIRED
#END CENTROID-CHANGES REQUIRED
```

Example:

```
# CENTROID-CHANGES
 Version-number: 1.0
 Start-time: 197001010000
 End-time: 199503012336
 Server-handle: BUNYIP01
# BEGIN TEMPLATE
 Template: USER
 Any-field: TRUE
# BEGIN FIELD
 Field: Name
 Data: Patrik
-Faltstrom
-Malin
-Linnerborg
#END FIELD
#BEGIN FIELD
 Field: Email
 Data: paf@bunyip.com
-malin.linnerborg@paf.se
# END FIELD
# END TEMPLATE
# END CENTROID-CHANGES
```

6.4 QUERY and POLLEES responses

The response to a QUERY command is done in WHOIS++ format.

6.5. Query referral

When referrals are included in the body of a response to a query, each referral is listed in a separate SERVER-TO-ASK block as shown below.

```
# SERVER-TO-ASK
 Version-number: // version number of index software, used to insure
                 // compatibility
 Body-of-Query: // the original query goes here
 Server-Handle: // WHOIS++ handle of the referred server
 Host-Name: // DNS name or IP address of the referred server
 Port-Number: // Port number to which to connect, if different from the
                // WHOIS++ port number

# END
```

Required/Optional table

Version-number REQUIRED, value should be 1.0
Body-of-query OPTIONAL
Server-Handle REQUIRED
Host-Name REQUIRED
Port-Number OPTIONAL, must be used if different from port 63

Example:

```
# SERVER-TO-ASK
 Version-Number: 1.0
 Server-Handle: SUNETSE01
 Host-Name: sunic.sunet.se
 Port-Number: 63
# END
```

7: Reply Codes

In addition to the reply codes listed in [Deutsch 95] for the basic WHOIS++ client/server interaction, the following reply codes are used in version 1.0 of this protocol.

113 Requested method not available	Unable to provide a requested compression method. Contacted server will send requested data in different format.
227 Update request acknowledged	A DATA-CHANGED transmission has been accepted and logged for further action.

503 Required attribute missing	A REQUIRED attribute is missing in an interaction.
504 Desired server unreachable	The desired server is unreachable.
505 Desired server unavailable	The desired server fails to respond to requests, but host is still reachable.

8. References

[Deutsch 95] Deutsch, et al., "Architecture of the WHOIS++ service", RFC 1835, August 1995.

[Faltstrom 95] Faltstrom, P., et al., "How to Interact with a WHOIS++ Mesh, RFC 1914, February 1996.

9. Security Considerations

Security issues are not discussed in this memo.

10. Authors' Addresses

Chris Weider
Bunyip Information Systems, Inc.
310 St. Catherine St. West
Montreal, PQ H2X 2A1
CANADA

Phone: +1-514-875-8611
Fax: +1-514-875-6134
EMail: clw@bunyip.com

Jim Fullton
MCNC Center for Communications
Post Office Box 12889
3021 Cornwallis Road
Research Triangle Park
North Carolina 27709-2889

Phone: 410-795-5422
Fax: 410-795-5422
EMail: fullton@cnidr.org

Simon Spero
EMail: ses@eit.com