Network Working Group
Request for Comments: 1914
Category: Standards Track

P. Faltstrom
Bunyip Information Systems, Inc.
R. Schoultz
KTHNOC
C. Weider
Bunyip Information Systems, Inc.

February 1996

How to Interact with a Whois++ Mesh

Status of this Memo

1. Overview

In the Whois++ architecture [Deutsch94],[Weider94], mesh traversal is done by the client, since each server ’refers’ the client to the next appropriate server(s). The protocol is simple. The client opens a connection to a server, sends a query, receives a reply, closes the connection, and after parsing the response the client decides which server to contact next, if necessary.

So, the client needs to have an algorithm to follow when it interacts with the Whois++ mesh so that referral loops can be detected, cost is minimised, and appropriate servers are rapidly and effectively contacted.

2. Basic functionality

Each Whois++ client should be configured to automatically send queries to a specific Whois++ server. The deault Whois++ server can vary depending on which template is desired, and the location of the client with respect to the WHOIS++ index mesh, but as a rule the server should be as local as possible.

```
                        A
                       / \
                      B   C
                     / \   \
           Z -----> D   E   F
                   / \
                  G   H
```

Fig 1: The client Z is configured to first query server D

After getting responses from a server, the client can act in several ways. If the number of hits is greater than zero, the response is just presented to the user. If the client gets one or many servers-to-ask answers, the client should be able to automatically resolve these pointers, i.e. query these servers in turn.

```
                        A
                       / \
                      B   C
                     / \   \
           Z <----- D   E   F
             \     / \
              --> G   H
```

Fig 2: The client Z gets a "servers-to-ask G" response from D and therefore may automatically queries server G.

3. How to navigate in the mesh

A client can use several different strategies when traversing or navigating around in the mesh. The automatic way of doing this is to just "expand the search" (described in 3.1) and a second method is to use the "Directory of Servers" (described in 3.2).

3.1. Expansion of searches

If the number of hits is zero, or if the user in some way wants to expand the search, it is recommended for the client to issue a ’polled-by’ and ’polled-for’ query to the server. The client can then repeat the original query to the new servers indicated.

```
                        A
                       / \
              /-----> B   C
             /       / \   \
           Z <----- D   E   F
                   / \
                  G   H
```

Fig 3: The client Z gets a "polled-by B" response from D and therefore queries server B.

The client must always keep track of which servers it has queried because it must itself detect loops in the mesh by not querying the same server more than once.

```
                        A
                       / \
                   /- B   C
                  /  / \   \
           Z <---/  D   E   F
                   / \
                  G   H
```

Fig 4: The client Z gets a "servers-to-ask D" response from B but Z does not query D because the server D has already been queried.

So, the default expansion of a query by a client causes increasingly more comprenhensive index servers to be queried; the forward knowledge contained in the index server mesh allows rapid pruning of these larger trees.

All loop detection and elimination is done in the client, rather than in the server mesh. This decision was made because loop detection and elimination are quite difficult to build into the mesh if we are to continue to allow each server to participate in multiple hierarchies within the mesh.

3.1.1. Optimising the mesh

If organization A tends to use organization B’s WHOIS++ server frequently, for example if A is cooperating in a project with B, A may wish to make B’s server locally available by creating a local index server which retrieves the centroid for both organizations. When A’s client then expands a query which is looking for someone at B, the client can much more rapidly resolve the query, as it does not have to find the top level servers for the tree to which A and B both belong.

```
                        A
                       / \
                      B   C
                     / \   \
           Z        D   --> F
                   / \
                  G   H
```

           Fig 5: The server B gets a centroid from server F

```
                        A
                       / \
                      B   C
                     / \   \
           Z <----> D   --- F
                   / \
                  G   H
```

  Fig 6: The client queries server D, gets zero hits back, expands the
             search and gets a "polled-by B" response back.

```
                        A
                       / \
                 /--> B   C
                /    / \   \
           Z <-/    D   --- F
                   / \
                  G   H
```

    Fig 7: The client Z queries server B and gets "servers-to-ask F"
                             response back.

```
                        A
                       / \
                      B   C
                     / \   \
                    D   --- F <-----> Z
                   / \
                  G   H
```

       Fig 8: The client Z queries server F and gets the answer.

The example given in Fig 5-8 shows that the algorithm works even though the Whois++ mesh is not a tree. There are many reasons why a given index server mesh might be 'short-circuited'. For example, in the case of a multinational company, the Swedish branch of Acme Inc., is polled both by the national server in Sweden and the headquarters server in the USA. By querying the Swedish server, one finds all persons working at the Swedish branch of Acme Inc., but by querying the Acme Inc. server in the USA, you will find all employees in the company, including those in Sweden.

Note that the location of a server does not implicitly narrow the search, i.e. you have to specify all information when sending a query to a server. In the example above, one can see that by just querying a server for companies in the USA, you will not implicitly only get hits from records in the states, because the Acme Inc. server in the states has polled a server in Sweden. So, in this case you have to explicitly include "country=USA" in the query if you are only interested in those records.

Although the WHOIS++ index service has been designed to make searches at any location in the index mesh quite effective and efficient, blindly expanding the query can incur an exponentially growing cost in resources, and, as charging for responses is implemented in parts of the WHOIS++ index service mesh, growing cost, automatic expansion is not recommended. More sophisticated clients should also be configurable to "cut off" some servers from a search, i.e. a blacklist of servers. This might be needed when searching for records and one server might have a very high cost (in dollars) so one might want to explicitly forbid the client to send queries to that server.

3.1.2. The algorithm used by the client

By following this algorithm a client finds all records in a mesh which the first Whois++ server queried belongs to.

The algorithm for the client follows:

```
Query := data to search for;
QueriedServers := {};
AnswerList := {};
OriginalServers := { known servers to this client };
while OriginalServers is not empty do:
      ServerList = OriginalServers;
      while ServerList is not empty do:
            Server := ServerList[1];
            if Server is not in QueriedServers then do:
                  send Query to Server;
                  Answer := answer from Server;
                  append ServersToAsk to ServerList;
                  remove Server from ServerList;
                  append Answers to AnswerList;
            end;
      done;
      if query should be expanded then do:
```

```
                  ServerList := OriginalServers;
                  OriginalServers := {};
                  while ServerList is not empty do:
                        Server := ServerList[1];
                        send Polled-For-Query to Server;
                        Answer := answer from Server;
                        append Answer to OriginalServers;
                        remove Server from ServerList;
                  end;
            done;
      done;
      display AnswerList to user;
```

3.2. The Directory of Servers

A second way of finding the correct server to query is to use a separate service we call the Directory of Servers. The Directory of Servers is a special Whois++ server which polls every Whois++ server for information about common information among the records on that perticular server.

3.2.1. How should a client use the Directory of Servers?

A client that want to very quickly find what servers serves USER templates in Sweden, should do it this way:

1) The hostname and portnumber of the directory of Servers have to be preconfigured in the current version of the protocol.

2) Query the Directory of Servers for serverhandle records for country sweden. This gives information of all these servers. By presenting this information to the user the user should be able to start the search at some closer server.

Note that we at this moment doesn’t think this should be an autmatic process in the client. The Directory of Servers should be used for giving the user information about what Whois++ servers that exists.

In the future a technique might have developed that makes it possible for a client to do this selection automatically depending on the query the user issues.

3.2.2. What does the serverhandle record look like?

The attributes that must be in all serverhandle records are:

```
Server-Handle: The handle for this server.
Host-Name:     The (current) hostname of this server.
Host-Port:     The (current) portnumber for this server.
```

Part from that information, the record can include other attributes like:

```
Admin-Name:        Patrik Faltstrom
Admin-Email:       paf@bunyip.com
Admin-Phone:       +1-514-875-8611
Organization-Name: Bunyip Information Systems Inc.
Description:       USER information
Menu-Item:         World (Bunyip Information Systems inc)
City:              Montreal
State:             Quebec
Country:           Canada
:
:
(Other attributes that can identify all records on this server, for
example domainname)
```

The information in the Navigation record is intended to be presented to a user.

3.2.3. Example

An example of how an interaction with the Directory of Servers is done follows. The characters '<' and '>' displays if it is the client ('<') or responding server ('>') which is responsible for the output:

```
> % 220-This is services.bunyip.com running Bunyip-Whois++: DIGGER 1.0.5
> % 220 Ready to go!
< template=serverhandle and bunyip
> % 200 Search is executing
> # FULL SERVERHANDLE BUNYIPCOM01 BUNYIPCOM01
>  SERVER-HANDLE: BUNYIPCOM01
>  HOST-NAME: services.bunyip.com
>  HOST-PORT: 63
>  ADMIN-NAME: Patrik Faltstrom
>  ADMIN-EMAIL: paf@bunyip.com
>  ORGANIZATION-NAME: Bunyip Information Systems Inc.
>  DESCRIPTION: USER information
>  DESCRIPTION: Directory of Servers
>  DESCRIPTION: Toplevel Index server in the world
>  MENU-ITEM: World (Bunyip Information Systems inc)
>  CITY: Montreal
>  COUNTRY: Canada
> # END
>
> # FULL SERVERHANDLE BUNYIPCOM01 BUNYIPCOM02
>  SERVER-HANDLE: BUNYIPCOM02
>  HOST-NAME: services.bunyip.com
>  HOST-PORT: 7778
>  ADMIN-NAME: Patrik Faltstrom
>  ADMIN-EMAIL: paf@bunyip.com
>  ORGANIZATION-NAME: Bunyip Information Systems Inc.
>  DESCRIPTION: USER information
>  MENU-ITEM: Bunyip Information Systems
>  CITY: Montreal
>  COUNTRY: Canada
> # END
>
> % 226 Transaction complete
> % 203 Bye, bye
```

4. Caching

A client can cache all information it gets from a server for some time. For example records, IP-addresses of Whois++ servers, the Directory of Services server etc.

A client can itself choose for how long it should cache the information.

The IP-address of the Directory of Services server might not change for a day or two, and neither might any other information.

4.1. Caching a Whois++ servers hostname

An example of cached information that might change is the chached hostname, IP-address and portnumber which a client gets back in a servers-to-ask response. That information is cached in the server since the last poll, which might occurred several weeks ago. Therefore, when such a connection fails, the client should fall back to use the serverhandle insted, which means that it contacts the Directory of Services server and queries for a server with that serverhandle. By doing this, the client should always get the last known hostname.

An algorithm for this might be:

```
response := servers-to-ask response from server A
IP-address := find ip-address for response.hostname in DNS
connect to ip-address at port response.portnumber
if connection fails {
   connect to Directory of Services server
   query for host with serverhandle response.serverhandle
   response := response from Directory of Services server
   IP-address := find ip-address for response.hostname in DNS
   connect to ip-address at port response.portnumber
   if connection fails {
       exit with error message
   }
 }
 Query this new server
```

5. Security Considerations

Security considerations when using the Whois++ protocol is described in [Deutsch94].

A client should be able to have a "blacklist" of servers it should not query, because it might happen that fake Whois++ servers is put up on the net. When such a fake Whois++ servers is found, a user should be able to configure it’s client to never query this server.

Note that a client should be careful when expanding a query by either using normal expansion or using the directory of servers. A query might take a long time, so a user should be able to quit in the middle of such a transaction. This is though more a question of user interaction than a plain security issue.

6. References

[Deutsch94] Deutsch P., Schoultz R., Faltstrom P., and C. Weider, "Architecture of the Whois++ service", RFC 1835, August 1995.

[Weider94]   Weider C., Fullton J., and S. Spero, "Architecture of the WHOIS++ Index Service", RFC 1913, February 1996.

7. Authors' Addresses

Patrik Faltstrom
BUNYIP INFORMATION SYSTEMS, inc
310 St Catherine St West, Suite 300
Montreal, Quebec
CANADA H2X 2A1

EMail: paf@bunyip.com

Rickard Schoultz
KTHNOC, SUNET/NORDUnet/Ebone Operations Centre
S-100 44 STOCKHOLM
SWEDEN

EMail: schoultz@sunet.se

Chris Weider
BUNYIP INFORMATION SYSTEMS, inc
310 St Catherine St West, Suite 300
Montreal, Quebec
CANADA H2X 2A1

EMail: clw@bunyip.com