Carlos Bueno
May 2004

A Distributed Web Search Protocol -- Dowser/0.1
draft-dowser-spec-00.txt

Status of This Memo

Abstract

Dowser is an application-level, peer-to-peer protocol for creating a searchable index and cache of documents. It is intended for both small-scale intranets and Worldwide Web-scale indexes. This document describes the messages that members, or "nodes", of the network pass to each other to distribute workload, request & respond to queries, and maintain the index.

Table of contents

1. Introduction

1.1 Purpose

The express purpose of the Dowser protocol is to encourage open research into web-scale, decentralized indexing systems. The specification for the Hypertext Transfer Protocol [RFC1945] has this observation:

> Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods to be used to indicate the purpose of a request.

HTTP is in widespread use by hundreds of millions of people every day, and they issue hundreds of millions of searches for information. Most of these searches are served by centralized "search engines" that cache copies of as many websites as possible to create their indexes. The problems of scale have so far been admirably met [GOOGLE], but the operating costs of web-scale engines are now out of the reach of most Universities and corporations. Conversely, larger and larger amounts of storage and processing power are available to personal computer users every year.

Dowser is an extension to HTTP that can be used by itself or added to existing HTTP implementations. Each node on the network claims a small part of the keyspace of a distributed hash table [CHORD]. Indexes of documents are keyed under the hash of the word; documents themselves are keyed by their Uniform Resource Identifier [URI]. There is also a "cache of caches", to allow popular documents or indexes to be retrieved from any node that has a copy.

The syntax and formatting rules are inherited from HTTP/1.1 [RFC2616]. The required "Host" header in HTTP/1.1 does not really apply to Dowser, but including it shouldn't hurt anyone. Several other headers and return codes are adopted for use.

1.2 Definitions

node

> In this protocol there is no difference between a "client" or a "server". They are all equal "nodes" on the network, capable of sending and serving requests. Each node is identified by a 40-digit hexadecimal number called a "node-id". They participate in creating, storing and serving a distributed hash table. This hash table can contain document caches, indexes of those documents, and other data.

For sanity's sake, the term "server" can be taken to mean "responding node" and "client" to mean "requesting node" in the context of the conversation being described.

hash, key

Hash functions are used to evenly distribute data around the network and to uniquely identify nodes and pieces of information. Dowser uses the Secure Hash Algorithim, version 2 [SHA]. The terms "hash" and "key" refer to the 40-digit hexadecimal number gotten from passing some piece of information through the SHA function, e.g.:

SHA("foo") == 0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33

distributed hash table

A distributed hash table is a key-->value mapping that is contained on many computers, where they keys are hashed.

range

The "range" of a node is defined by the first and last key of the hash table it is expected to store data under.

left-hand, right-hand

The total keyspace is imagined as the perimeter of a circle, or a "ring". While standing in the center, points to the left generally decrease and points to the right generally increase, except at the point where "fff..." meets "000...".

A useful analog is timezones, where West is left and East is right. The time is always earlier to the left except at the International Date Line.

1.3 URL vs. URI vs. URN

The protocol descibed here has a focus different from most filesharing protocols. Searches for files are not broadcast but targeted; nodes are authoritative regarding the location and checksums of data in their ranges. Searches for a particular node are sent to arbitrary nodes but not broadcast per se; they are not forwarded but responded to directly.

In most ad-hoc filesharing protocols the only unique identifier is the checksum. The filesystem names and metadata may be different for different copies but the checksum remains the same. In HTTP jargon this is a Universal Resource Name or URN , that which uniquely identifies the file without necessarily referencing its location or common name [RFC2396].

The content of a cached web page may change with time. This means the checksum will change while its Univeral Resource Locator, the origin of the page, remains constant.

Indexes built from these cached pages are also treated as files, one index file per word or phrase. An index's Universal Resource Name is

that word. (More correctly it would be "dowser://foo", not "foo".)

Searching in Dowser therefore starts with a lookup for the node(s) who handle the search term(s). The search terms (URN) are then sent to those nodes, resulting in either a) the content of index file or b) the latest checksum of the index file and a list of nodes that have previously downloaded a copy. Retrieving a cache of a web page is done the same way.

1.4 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [34].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

In reality, the complications of this protocol are mostly in the state information, not the ins and outs of the messages being passed.

2. Some Examples

We will assume the reader is familiar with HTTP transactions. It is probably most instructive to give examples using the new methods and then a discussion of particulars. Elipses ("...") are used to truncate long hashes for readability.

2.1 NODEFIND

NODEFINDs are the bread and butter of routing between nodes. The client wants to find the node(s) that have items under a certain hash key, in this example, "5c89379d0aa9840ac910fd8cacbde2dbf877214a". This can be a search term, url, or anything. The responding node may or may not have this key. If not, it tells the client about nodes it knows about that are closer to the target.

```
NODEFIND 5c89379d0aa9840ac910fd8cacbde2dbf877214a Dowser/0.1
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 12222... a5754...
```

```
      Last-key: 13333...
      Port: 4666
```

```
   - Response -
```

```
   Dowser/0.1 310 Not in my range
   Ring-Id: deadbeef0000000000000000000000000000000
   Node-Id: 48888...  de34....
   Last-key: 49999...

   192.0.2.10 8000 50000f... 599999...
   192.0.2.20 6876 43333a... 610000...
```

- Or -

```
   Dowser/0.1 211 That's me
   Ring-Id: deadbeef0000000000000000000000000000000
   Node-Id: 4000... de34....
   Last-key: 7aaaa...
```

There is a lot to cover here. The first line of the request is like HTTP, identifying the protocol and version, the 'method' or action to be done, and the 'path' or key being requested. There are three additional headers which MUST be included in any message: Ring-id, Node-id, and Last-key. Additionally, the Port header MUST be included in every request.

2.1.1 Node-id & seed

The node-id is comprised of two 40-digit hexadecimal numbers, separated by whitespace. The first number serves as the FIRST key in this node's range. This number MUST be generated by hashing some random data iteratively. To help ensure this has occurred, the second number (the "seed") MUST be the second-to-last result of the iterations.

```
      Node-id: b274f2e2a8d2881035af5866014e9ad5510ab15d
         cdd2ae2594a83ef90c05ee6014b78631db8538d8
```

This example (linewrapped to fit) is well-formed because the first number is the SHA hash of the second number. Servers MUST check that the node-id and its "seed" are correct. If not, they MUST send a 412 "Precondition failed" error.

2.1.2 Last-key

Last-key is also an SHA hash, and is the LAST key in this node's range. The last-key may change over time, as nodes leave or join the

network.

Note: it has been brought up that a node might be able to route a large portion of traffic to itself by making its Last-key equal to the node-id minus 1. In Internet Protocol terms, this would be similar to setting your network address to 255.255.255.255. There is

nothing in the protocol to prevent this, but implementations should come up with a way to deal with it reasonably. Our feeling that it is self-correcting. If someone tries it on a network with a lot of nodes one hopes he or she has good fire-supression equipment handy; but the concern is the possible damage to the network routing.

2.1.3 Ring-id

Ring-id is yet another SHA hash to identify the group of nodes it wishes to communicate with. This allows many Dowser networks to be running on an internet at once. Alternate rings are useful for testing implementations of the protocol on a large scale, or for partitioning networks in general.

The "official" public ring-id:

```
Dowser/0.1  -->  deadbeef00000000000000000000000000000000
```

If the Ring-id header sent does not match the ring-id of the server, it MUST send a 412 "Precondition falied" message, and may provide information in the body of the response as to the reason. Note: there is a security risk in actually revealing the server's ring-id in the response, if the ring is intended for private data.

2.1.4 Port

Simple enough: we need to advertise what TCP or UDP port we are listening on for future reference.

2.1.5 Server Responses

If the hash is not in the listening node's range, it returns a 310 code and a list of nodes that are closer to the target. The list of nodes takes this form:

```
IP PORT NODE-ID LAST-KEY
```

If it is in range, it sends a 211 "That's Me" code.

When overlap occurs a node may choose which one to contact first in an implementation-dependent way, such as IP or network distance, or closeness of the desired key to the node-id.

Note: IPv4 addresses are used in these examples, but implementations SHOULD be written to handle IPv6 addresses as well, e.g.:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
1080:0:0:0:8:800:200C:4171
3ffe:2a00:100:7031::1
1080::8:800:200C:417A
::192.9.5.5
```

```
::FFFF:129.144.52.38
2010:836B:4179::836B:4179
```

2.2 SEARCH

We have the basis of all conversations, and have described how nodes identify themselves and each other. Now we can go to the next level: content routing.

- Request -

```
SEARCH foo+bar+baz Dowser/0.1
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 1d3470c... 55754...
Last-key: 1e400cc...
Port:  4666
```

- Response -

```
Dowser/0.1  200 OK
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 48888... de34...
Last-key: 49999...
Expires: 864000
Content-key: 3fdb13677b10691debb3909dd917b00ee751115a

URL TITLE AGE SNIPPET [SNIPPET [SNIPPET [...]]]
URL TITLE AGE SNIPPET [SNIPPET [SNIPPET [...]]]
...
```

- Or -

```
Dowser/0.1 300 Multiple choices
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 48888... de34...
Last-key: 49999...
Expires: 864000
Content-key: 3fdb13677b10691debb3909dd917b00ee751115a

192.0.2.10 8000 50000f... 599999...
192.0.2.20 6876 43333a... 610000...
```

A SEARCH request is almost exactly like a nodefind except the "path" is a list of search terms. The server determines if any of the terms

fall into its range. How to break up the terms may be implementation-specific, but the simplest is to break on whitespace.

If any are in-range it returns a 300 code, the "content-key" of the data (a SHA checksum of the content) a list of the nodes that have

requested that data.

Or, it may return the index data itself. Requesting nodes SHOULD keep a copy of the data it receives for a reasonable amount of time, as hinted by the Expires: header. The data should be keyed under the content-key, for retrieval via a CACHE request. This helps distribute the workload. The index is in the form:

URL TITLE AGE SNIPPET [SNIPPET [SNIPPET [...]]]

And is tab-delimited. AGE is the number of seconds since the record was created. One or more "SNIPPET"s contain text that form an abstract of the document.

As always, the server can alternately send the standard 310 Not in Range, or some error code.

2.2.1 Expires:

Indexes and caches of web documents are not long-lived in the way a media file in traditional peer-to-peer networks are. The Expires header hints at how long a node should cache this data, expressed in seconds from the present time.

2.2.2 Content-key:

This is a SHA hash of the data's content, used to positively identify it. This is different from the hash of the URL or search term. A page whose URL is

"http://foo.example.com"

... and whose content is

"<h1>Foo!</h1>"

... would have a URL hash of cfab46bb7dbd11e6187360d429586e2942f2d42e and a content-key of cf5ce65061218164e4148038cc3a56a9e988fe7a. The URL hash is a unique identifier of the document's origin, while the content-key is an identifier of a particular version of that document.

2.2.3 Search syntax

There are a few schemes for search syntax out in the wild. The most

common has operands for words that must or must not appear, and exact phrases. Implementations MUST honor these operands to the best of their ability.

foo +bar

Means that the documents MUST contain "bar" and MAY contain "foo"

```
foo -bar
```

Means that the documents MUST contain "foo" and MUST NOT contain "bar". When there is only one term without a "-" operand, the "+" is implied.

```
"foo bar"
'foo bar'
```

Means that the documents MUST contain the exact phrase "foo bar", in that order.

Note: in the SEARCH example in section 2.0, it may seem like the "+" signs in "foo+bar+baz" are search syntax. They are not; the terms are "url-encoded", meaning spaces are encoded as "+" and other non-word characters are encoded as %HEX where HEX is the hexadecimal ASCII code. Therefore "foo +bar" would appear as "foo+%2Bbar"

Implementations may honor other operands as they see fit.

2.3. URL caching

Like a SEARCH, a URLCACHE does not necessarily return the data -- it can return the content-hash of the data and a list of the last X nodes to request that data, where X is implementation-specific. If none of the listed nodes have the data, the requesting node may then request it from the original node.

- Request -

```
URLCACHE fb9642989a2222305832a5d9c29b34... Dowser/0.1
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 1d3470c... a5754......
Last-key: 1e400...
Url:  http://foo.example.com/some/page.html
Port:  4666
```

- Response -

```
Dowser/0.1  300 Multiple choices
Ring-Id: deadbeef00000000000000000000000000000000
Node-Id: 48888...  de34...
Last-key: 49999...
Expires: 864000
```

```
      Content-key: 3fdb13677b10691debb3909dd917b00ee751115a

      192.0.2.10 8000 50000f... 599999...
      192.0.2.20 6876 43333a... 610000...
```

   - Or -

```
Dowser/0.1  200 OK
Ring-Id: deadbeef0000000000000000000000000000000
Node-Id: 48888...  de34...
Last-key: 49999...
Expires: 864000
Content-key: 3fdb13677b10691debb3909dd917b00ee751115a

... (file data) ...
```

2.4. Content caching

This can be page caches, indexes, or really anything. A client sends a request with the content-key as the path:

- Request -

```
CACHE 3fdb13677b10691debb3909dd917b00ee751115a Dowser/0.1
Ring-Id: deadbeef0000000000000000000000000000000
Node-Id: 1d3470c...  de34...
Last-key: 1e400...
Url:  http://foo.example.com/some/page.html
Port:  4666
```

- Response -

```
Dowser/0.1  200 OK
Ring-Id: deadbeef0000000000000000000000000000000
Node-Id: 48888...  de34...
Last-key: 49999...
Expires: 864000
Content-key: 3fdb13677b10691debb3909dd917b00ee751115a

... (file data) ...
```

- Or -

```
Dowser/0.1 404 not found
Ring-Id: deadbeef0000000000000000000000000000000
Node-Id: 48888...   de34...
Last-key: 49999...
```

todo: content-length? simultaneous downloading? the Range: & Content-length: headers?

2.5. CRAWL

So we know how to find nodes and content. The last question is: where does the content come from? And the indexes?

```
   CRAWL http://foo.example.com/foo/bar/baz.html Dowser/0.1
   Ring-Id: deadbeef00000000000000000000000000000000
   Node-Id: 12222... a5754......
   Last-key: 13333...
   Referer: http://foo.example.com/foo/index.html
   Port: 4666
   Expires: 864000
   Content-key: 3fdb13677b10691debb3909dd917b00ee751115a
```

- Response -

```
   Dowser/0.1 202 Accepted
   Ring-Id: deadbeef00000000000000000000000000000000
   Node-Id: 48888...   de34...
   Last-key: 49999...
```

- Or -

```
   Dowser/0.1 310 Not in my range
   Ring-Id: deadbeef00000000000000000000000000000000
   Node-Id: 48888...   de34...
   Last-key: 49999...

   192.0.2.10 8000 50000f... 599999...
   192.0.2.20 6876 43333a... 610000...
```

When a node crawls a page, it usually has links on it. If the hash of those second-level URLs are not in the node's range, it sends a CRAWL annoucement to the nodes that do handle that range, so they can continue the crawl.

A node SHOULD first check (via URLCACHE) if a cached copy exists before downloading a page directly.

Implementations SHOULD respect the "robots.txt" protocol for nice spiders.

##todo: so where do we START crawling? that is not part of the protocol because it could be a network of office machines sharing their spreadsheets, or whatever -- need to say that in a nice way.

2.6.INDEXADD

An INDEXADD is an informational message to a node that a particular URL contains a keyword inside the server's range.

```
INDEXADD 09dd917b00ee751115a3fdb13677b10691debb39 Dowser/0.1
Ring-Id: deadbeef0000000000000000000000000000000
```

```
      Node-Id: 12222...  a5754......
      Last-key: 13333...
      Term:  foo bar baz
      Url: http://foo.example.com/foo/bar/baz.html
      Expires: 864000
      Content-key: 3fdb13677b10691debb3909dd917b00ee751115a
      Port: 4666

##todo: positions, counts, tdf*idf?
```

- Response -

```
      Dowser/0.1 202 Accepted
      Ring-Id: deadbeef00000000000000000000000000000000
      Node-Id: 48888...  de34..
      Last-key: 49999...
```

- Or -

```
      Dowser/0.1 310 Not in my range
      Ring-Id: deadbeef00000000000000000000000000000000
      Node-Id: 48888...  de34..
      Last-key: 49999...

      192.0.2.10 8000 50000f... 599999...
      192.0.2.20 6876 43333a... 610000...
```

The Term: header contains what words the server may be interested in, separated by tabs. An INDEXADD message implies that the requesting node has a copy of the document. Listening nodes should retrieve it from that node via a CONTENTCACHE message. This puts a cost on adding words to the index.

3. Announcing

When a new node joins the network, it must first be provided with a list of "well-known" nodes and a ring-id. Knowing nothing else about the network topology, it generates a node-id and sends a NODEFIND request on that key to those well-known nodes for its own key:

```
      NODEFIND 12222...0 Dowser/0.1
      Ring-Id: deadbeef00000000000000000000000000000000
      Node-Id: 12222... a5754...
      Last-key: 12222...
      Port:  4666
```

This eventually leads the new node to its "neighbors" on the ring. The new node's range SHOULD initially be of the same length as its left-hand neighbor, but MAY grow or shrink according to how much capacity it has at hand, but the Last-key MUST NOT be less than the node-id of its closest neighbor.

4. Redundancy

All nodes may not be available at all times, and the "overlap" is not guaranteed, requiring some fiat redundancy. So:

Each node assumes a range starting with its nodeid and That defines its "core" range. It SHALL also handle an "auxilary" range of equal length, calculated by adding 8 to the first digit. The addtion does not carry. For example:

Node A "Core" range:

```
9000000000000000000000000000000000000000 to
c000000000000000000000000000000000000000
```

Auxilary range:

```
1000000000000000000000000000000000000000 to
4000000000000000000000000000000000000000
```

5. Optional headers

To save on bandwidth, nodes MAY compress their responses. A requesting node MAY indicate that it can accept compressed data with this optional header:

```
Accept-Encoding: gzip, deflate
```

If a server compresses a response, it MUST inidcate that with this header:

```
Content-encoding: gzip
```

The server MAY also send this optional header to SEARCH responses to help guide the user:

```
Related-terms: meta, syntax, jargon
```

6. Error conditions

Many error codes from HTTP/1.1 apply to Dowser.

500 Internal Server error

501 Not Implemented

503 Server busy (Service unavailable)

505 HTTP Version Not Supported

400 Bad request

404 Not Found

412 Precondition failed

When a node's id and seed don't check out, for instance. This is different from a mal-formed request which should be responded to with a 400. Also, nodes may keep a "blacklist" of node-ids and/or IPs that they do not wish to do business with (See SPAM). In the case of blacklists, the server MAY simply close the connection, but it's polite to give some error information.

Implementations MAY elect to withold the Ring-id header in this case, for security.

7. Notes on implementation

A reference implementation is currently in alpha testing and should be released by mid-summer 2004 [IMPL]

7.1 Ranking

There is not a lot of ranking data returned by SEARCH requests. The idea is to return somewhat raw indexes to the client and let it sort things out through intersecting multiple responses and its own biases.

One time-saving trick may be employed by implementations: recall in the SEARCH example, all of the search terms were sent, even though it's unlikely one node will handle all of them. The reason is to allow nodes to pick out terms that tend to accompany the terms they are responsible for. The node may then keep an extra index of those accompanying terms and use it to trim and/or rank the results sent back.

7.2 Proxying

If a relatively new node receives a request for a key that is in its range but not its datastore, it may create a new request to its upstream neighbor for it, on the theory that it may still be in that node's datastore. It then keeps a copy while forwarding it to the

original requesting node. This allows new nodes to build on the work of older ones.

Implementations might want to work out semantics for long-lived connections on the lines of HTTP/1.1 Keep-alive headers.

7.3 TCP vs UDP

Several methods (NODEFIND, CRAWL, INDEXADD) have two interesting properties: they will be used often, and involve a very small amount of data. It is tempting to implement those parts over UDP and the rest over TCP, or to come up with some tricks to do it all over UDP.

7.4 Ping/Pong

Neighboring nodes should be in the habit of pinging their nearest neighbors, if they have not heard from them in a while. A "ping" can take the form of a NODEFIND using the neighbor's node-id.

7.5 MHTML

HTML documents of today tend to be comprised of many files including graphics, stylesheets, etc. It may be useful for implementations to offer "MHTML" versions of cached documents that contain these supporting files in one MIME-encoded file. If a client supports MHTML, it should include it in the Accept: header. Common-sense restrictions apply, such as not including executable scripts or plugins and files that do not come from the same domain.

7.6 Spam

The network described above may one day contain thousands of nodes. Thousands of nodes implies thousands of users looking for information, which nowadays means spam. An implementation may elect to deal with it in the way email clients do, with filters built from analysis of links classified by the user as "spam" or "useful". Even in the abscence of spurious traffic a good Bayesian filter may help tailor results.

A sophisticated method of injecting targeted spam involves generating node-ids until one pair is "close enough" to the hash of a desired keyword, where "close enough" is matching the first 4 or 5 digits. Even thousands of nodes will be sparse in the keyspace, so an exhaustive search is not necessary.

7.7 Indexing

A large sample of web documents suggests that there is a fairly constant number of terms, something on the close order of 14 million [GOOGLE].

An unpublished survey of a large sample of search engine queries suggests that there is also a fairly constant number of unique terms searched, on the close order of 1 million. This is very interesting.

Suppose we give more "weight" to words found in the query stream over those that are not. As we've seen in the adventures of Internet search engines over the last few years, documents often lie about their contents, but users rarely lie about their desires. They are vauge, perhaps, but not false.

. . . that is, until the query stream becomes a thing of value. A clever person might include a popular word and a made-up one into his or her web page, then send thousands of queries with both of those words to artificially boost the correlation. I have a wonderful proof describing how this gambit might be defeated, but unfortunately there is not enough room to include it here.

8. Acknowledgements

This specification builds on the years of work put into designing and implementing Dr. Berner-Lee's HTTP protocol by many thousands of people around the world. You know who you are.

Ideas and algorithims were stolen liberally from Chord, Circle, and Freenet. Lots and lots of ideas from the original distributed search projects, Harvest and WebAnts.

Special thanks to Richard Vasquez and Thomas Lackner for their devious minds and help with early implementations.

9. References

9.1 Informative

[GOOGLE] Brin & Page, "The Anatomy of a Search Engine", Stanford University, 1997

[CHORD] Stoica, et al. "A Scalable Peer-to-peer Lookup Service for Internet Applications", Sigcomm 2001 http://www.pdos.lcs.mit.edu/papers/chord:sigcomm01/

[IMPL] Dowser P2P client, http://dowser.sourceforge.net

9.2 Normative

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax and Semantics", RFC 2396, August 1998.

[HTTP] Feilding, et al. "Hypertext Transfer Protocol -- HTTP/1.1",

RFC2616, June 1999

[SHA] Eastlake & Jones, US Secure Hash Algorithm 1 (SHA1), RFC3174, September 2001

10. Security Considerations

10.1 Personal Information

Dowser deals directly with people's searching habits. While spreading this information around is kind of the point, implementations SHOULD take care not to "leak" any more than neccessary. For instance, it may be tempting to skip the NODEFIND sequence and just issue SEARCHes or URLCACHEs, since they also return node-finding information. This sends people's search terms and requested urls to nodes that don't strictly need it, and SHOULD be avoided.

Implementations can be expected to work closely with traditional HTTP applications, with their own privacy & security considerations.

10.2 Sensitive data

Dowser was originally intended to deal only with publically accessible information, but it can also be used for sharing sensitive data among trusted computers, say, all the word processor documents on an office LAN. For these situations, implementations should use some sort of IP whitelisting or subnetting and make the user aware of the risks.

11. IPR & Copyright

12. Contact Information

Carlos Bueno
6231 SW 78th St., Ste. 20
Miami, FL, USA 33143
Email: carlos@bueno.org

This document expires on November 4, 2004