FIND Working Group
INTERNET-DRAFT
<draft-ietf-find-new-cip-00.txt>

Jeff Allen
Bunyip Information
Systems, Inc.

Patrik Faltstrom
Tele2/Swipnet

The Common Indexing Protocol (CIP)

Status of this memo

This Internet Draft expires November 13, 1996.

Introduction

With the accelerating expansion of the availability of information of all kinds on the Internet, search technologies have become more and more important to users to help them turn the mountains of data into gems of information. Technologies such as WAIS, Z39.50, and the growth of ad-hoc interfaces based on HTML forms technology bring the task of searching a single database (locally or remotely) pretty much under control. A multitude of such databases and access protocols/languages exist, and are serving users satisfactorally every day.

In addition to the single databases available on the Net, we are also seeing an explosion of massive databases created by automated indexing tools, or (in rarer cases) by hand. Databases like Alta Vista, Webcrawler, and Yahoo are aggregating vast amounts of information into massive centrally maintained and accessed indexes of "everything out there". By all accounts, this approach to managing massive amount of widely distributed information is facing scaling problems (both those based on automated technology, and those based on human indexing), and is certainly not the end of the road for information retrieval technologies.

Distributed database technology would seem to be the next step. While much progress has been made with respect to managing searches in a distributed environment, for a variety of reasons, this technology has not come into use to make "searching the Internet" effective and efficient.

The Common Indexing Protocol (CIP) is proposed not only as a mechanism for distributing searches across several instances of a single type of search engine. The CIP approach was originally proposed in the context of enabling searches across multiple Internet white pages (people) database technologies (e.g., Whois++, X.500, PH) with a view to enabling the creation of a global directory of people. In this application domain, the raw information (personal data) is usually maintained by an organization to which the person is related (for purposes of data integrity and security). When it became clear that there was no one white pages technology that would be embraced by all organizations, the goal of a global directory service had to be met by a technology that would act across the diverse set of available systems.

With CIP, we plan to provide a scalable, flexible scheme to tie existing and future databases into distributed data warehouses which can scale gracefully with the growth of the Internet. A CIP mesh can be searched efficiently, and scales easily. The goal of this document is to define an architecture and protocol to move indexing information among existing and future servers. We will also explain how clients will make use of this information to conduct efficient searches.

Architecture and Protocol

In this section we define the architecture of the system, as well as the protocol used between conforming CIP servers. Finally, we describe the issues involved from a client's point of view, when executing a query.

Overview

* CIP integrates multiple data-access protocols

CIP is a protocol meant to be spoken between servers (existing and future) which provide a local database search. CIP is not a protocol for finding data. CIP allows servers to pass hints among themselves about where data is so that, in their native protocol, the servers can refer clients to the place(s) where their query will yield results. Due to these referrals, a query is "moved" towards parts of the distributed database where it is more likely to be successfully answered. This process is called query routing, and it's central to effective, efficient searching. CIP makes query routing possible.

* Indexing defined

Indexing, in the context of CIP, is the process of gathering some "reconnaissance information" which can be used to help route queries to the servers most likely to have answers. The process of indexing a body of data and passing that knowledge on to other parts of the distributed system is an investment, of sorts. By spending a little time behind the scenes distributing indexing information among themselves, a group of servers that form a distributed database can make sure that answering queries is only attempted by servers likely to have a productive part of the answer. Using query routing like this can avoid the wasteful floods of queries that must occur in a distributed system with no indexing capabilities.

The data generated and passed among these servers is called an index object. An index object is not the same as the full database. It is simply a set of "hints" on what kind of queries the index object originator will be able to answer. Based on the index object from a server, you cannot, in general, reconstruct that server's database. (See Security Considerations, below, for a discussion of why this is an important property for index objects to have.)

* Organization of index servers

A group of servers sharing index objects of a common type is called a "mesh". A mesh, as the name implies, can be more complicated than a simple tree. In general, it is simplest to think of the mesh as a rough tree-shaped hierarchy, with servers near the root of the tree having more knowledge about the rest of the tree than those nearer the leaves. Leaf nodes (called "data servers") know no indexing information, but they export index objects based on their data. Servers with some knowledge of the tree below them are called "index servers", since they hold index objects of their children, and can do query routing based on this knowledge.

The mesh is allowed to be less rigid than a strict tree because the organization of the mesh needs to remain fluid. Different types of data used for different applications may have divergent requirements for index structure. Because the mesh need no be a strict tree, these different structures may be overlayed on the same data servers. The mesh is a simple solution to a hard problem. It's impossible to decide on the "right" distributed architecture when faced with conflicting requirements, so CIP allows multiple concurrent indexing organizations.

* Implementation

CIP transactions take place on reliable data streams (currently, the only implementation is for TCP). CIP servers can reside on any TCP port, though the IANA assigned port number of XXX is common. It is

mostly an NVT-ASCII protocol, with the exception of the index objects themselves. The format for index objects is defined separately from CIP.

CIP's place in the World

Based on the initial definition of the system above, we'll present an example of what it means for CIP to be a backend protocol. Consider the following arrangement of a client, and two servers:

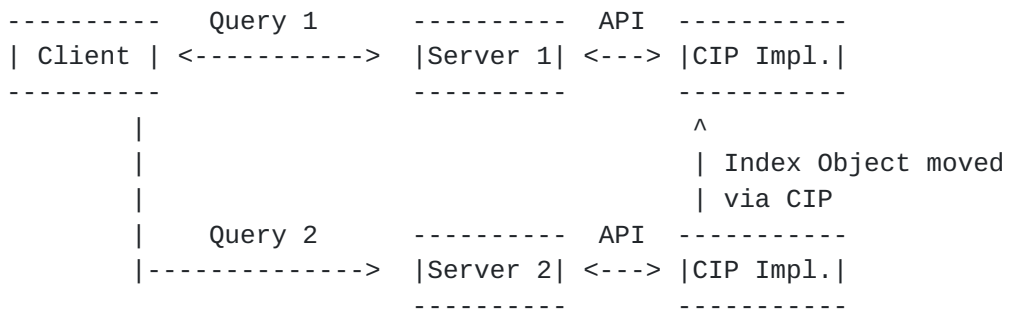

Figure 1.

In this example, we are dealing with a very small mesh, so small in fact, it consists of exactly two servers. Server 1 (S1) polls Server 2 (S2) via CIP. S1 holds not only all the data loaded into it by its administrator, but also a copy of the index object that S2 sent it.

The client begins a search by asking S1 for a particular item. This transaction (labeled "Query 1", above) takes place in the native data access language and protocol of S1. Server 1 could be a Whois++ server, an LDAP server, or some other server yet to be developed. The query could be for any kind of data appropriate for the server and its protocol.

During the course of executing the query, S1 checks both it's local data and the index object it has on hand from S2. Based on this search, it can generate a result immediately if it has the data locally, or it can generate a referral to the data if the index object it searches indicates the result may be on S2.

The client may or may not chase down the referral, depending on how it is programmed, or on what the user decides to do next. An important property of CIP is that the client side is responsible for tracking referrals. For clients which are not capable of understanding referrals (due to backwards in-compatibility), a proxy might be implemented on the server-side to chase referrals. However, this is not a CIP protocol issue; the proxy server would be specific to the native protocol in use on that server.

Note that the CIP implementation and the server implementation are in separate "boxes", even though they are not, as it might appear, separated into different processes. Instead, CIP is envisioned to be a service that can be added into a system with minimal modification. The work CIP does has been carefully abstracted from the work a conventional database server does so that the two can be connected in a straightforward manner via an implementation-defined API.

The CIP Dataset

For the purposes of making and managing CIP index objects, it is important to have a unique way to identify particular databases. Each database is called a dataset. A server may contain one or more datasets. Each dataset can be indexed in one or more ways.

To allow CIP servers to talk about datasets in a meaningful way, every dataset in the Internet which is indexed by CIP must have a network-wide unique dataset identifier, or DSI. To avoid requiring the IANA or other authority to worry abut trying to name every database on the net, we have chosen to use the ISO Object Identifier (OID) for DSI's. While OID's are a bit unwieldy to manage and understand by humans, they have a very easy to manage storage and transfer format. They are also essentially infinitely extensible, though certain limits on CIP protocol message sizes put practical limits on the OIDs to make the protocol easier to implement.

The part of the OID-space used by CIP is the same part used by SNMP enterprise numbers. Thus, to begin naming CIP datasets within an organization, it is necessary to get a enterprise number assigned to you by the IANA. For help on getting enterprise numbers, see [IANA].

In addition to describing datasets with OIDs, any place a DSI is given, a DSI description can be added. The description is a free-form text field intended to help humans understand the more computer-friendly OIDs.

The Index Object

Since not all data can be indexed the same way, CIP allows for many different kinds of index objects. All of them share a common container and general format. The exact derivation and specific contents of an index object is defined by the index policy, which is defined by CIP users who are interested in applying the protocol to a new domain of data. A domain of data is the set of all data with common needs with respect to indexing. Digital pictures, sound, and text would certainly fall into separate domains, since they have such vastly different needs with respect to searching. More subtly, full text databases and record-oriented databases might fall into different domains, depending on how one chose to index them. Because of this, a given dataset might

have one or more index objects associated with it; one index object for every domain.

A concrete example lies in the current use of CIP with Whois++. These servers exchange a form of index object called a centroid. A centroid is a list of tokens in which each token has associated with it the attribute and template type in which it appeared in the original data. An index object definition must include a specification of how the index is derived, and how it is formatted for transport. The Whois++ specification defines the derivation as the set of unique tokens (collated by attribute and template type) and the format as a nested set of attribute/value pairs. For an exact specification of the Whois++ centroid, see [RFC-1913].

When an index is passed between two servers, the domain is specified by the polling server. Thus, the server which is about to attempt to receive and make sense of the index object specifies what format it is expecting. If the sending server does not have an index object available for the specified domain, nothing will be transferred. In normal operation, this should not happen, since polling relationships are set up by server administrators, who should know what domains the potential peer provides index objects for. However, this mechanism leaves the door open for research on cross-domain transfer of indexing information. Note also, that it is possible within the existing protocol for a server to export index objects for two different domains based on the same data.

The index is packaged for transport in a container which looks like this:

```
    # INDEX-CHANGES
     Version-Number: 3.0
     Index-Type: <index object type-name>
     DSI: 1.3.6.1.4.1.<enterprise-number>.<local part of OID>
     DSI-Description: <ASCII string describing the dataset>
     Base-URI: <base url>
     Authentication-Type: Password
     Authentication-Data: xxxxxxxx
              ...                                ...
              ... domain-specific parts come here ...
              ...                                ...
    # END INDEX-CHANGES
```

There are two parts to the index container. The first is the domain-independent part, where meta-information defined by the core CIP protocol is stored.

The domain-independent information in the index object container includes:

Index-Type:
     This is the case-insensitive name of the index object type. The

only currently defined type name is "CENTROID". Experimental index object types should start with "X-". New type names are defined by the specifications written for new indexing objects.

DSI:
This is the DSI for the dataset from which this index object was created. See "The CIP Dataset" above for information on where DSIs come from.

DSI-Description: (optional)
A free-form human-readable line describing the dataset.

Base-URI: (optional)
This is a URI (Uniform Resource Identifier) which can be used (augmented by an actual search string) to find items in given index object. For a more complete description of the Base-URI and its use when preparing referrals, see "Navigating the Mesh" below.

After the domain-independent part comes the index object itself. The specific formatting of the index, as well as the rules governing how an index object is derived from a given dataset are defined separately from CIP as domain-specific extensions. The first existing object specification is for the Whois++ "centroid" index object type. See the "New Directions" for more information on future research into alternative domains and cross-domain index management.

In a future version of this Internet Draft, the issues involved in incremental updates will be addressed. There is still some contentiona as to whether it should be a domain-dependent or independent thing.

Updating the mesh - transferring an index

When a transfer of an index object is done, a mixture between a pushing and polling method is used. A server which has an index object that has changed must inform all of it's mesh neighbors that they should fetch an update to their current index objects. The command sent is called a Data-Changed command. It is only ever sent from the _polled_ server to a _polling_ server.

After this command is transferred, the polling server can choose when it wants to fetch the new index object. Existing implementations pick up the index object immediately, but there are provisions in the protocol to allow the polled server to suggest a low-traffic period during which it would prefer to handle the poll.

A polled server can send several Data-Changed commands to the same server even if that server have not fetched a new index. It is strongly recommend against Real-Time updates, i.e. it is best to not send a change notice for every change made in the data, and to instead accept a small amount of lag in the system in the name of lower polling overhead.

The Data-Changed command is only a _notification_ that a newer index object is available. Only once the polling server connects back to the polled server does the index object get transferred.

The Data-Changed command

The Data-Changed command includes the following data, for example:

```
# DATA-CHANGED
 Version-Number: 3.0
 Modification-Date: 199603041625
 DSI: 1.3.6.1.4.1.1375.1
 Host-Name: pollee-hostname
 Host-Port: 63
 Best-Time-Next-Poll: 199603050100
 Window-Size: 3600
# END
```

Notice that the modification-date and best-time-next-poll are both given in GMT, and the window-size in seconds. This window for polling is the recommended one from the polled servers point of view. It might be the polling server that have to, because of work-load, poll whenever it is possible.

After a Data-Changed command is sent from one server to another one, the command is acknowledged, and the connection is closed.

Polling

The Poll command is the command that is sent from the polling server to the polled server when it wants to request a new index. The Poll command is normally only sent as a response to one or more earlier Data-Changed commands sent to the polling server.

The format of the POLL command is as follows (an example):

```
# POLL
 Version-Number: 3.0
 Character-Set: UNICODE-1-1-UTF-8
 DSI: 1.3.6.4.1.1375.1
 DSI-Description: Bunyip
 Type-Of-Poll: CENTROID
 Tokenization-Type: Tokens
# END
```

The value of the type-of-poll attribute implicates some extra attributes, which in the case of CENTROID is the attribute Tokenization-Type, which in turn is the tokenization-type the polling server requests.

Navigating the mesh

With the CIP infrastructure in place to manage index objects, the only problem remaining is how to successfully use the indexing information to do efficient searches. CIP allows query routing, which is essentially a client activity. A client connect to one server, which redirect the query to servers with the answer. This redirection message is called a referral.

* The Referral

The concept of a referral and the mechanism for deciding when they should be issued is described by CIP. However, the referral itself must be transferred to the client in the native protocol, so its syntax is not directly a CIP issue. Recall the example in Figure 1. **The server S1 generates a referral, directing the client to contact** server S2 for more results. The mechanism for deciding this referral needed to be made resides in the CIP part of the server. The mechanism for generating and sending that referral to the client resides in the server itself.

A referral is made when a search against the index objects held by the server shows that there may be hits available in one of the datasets represented by those index objects. There may be no more than one referral per dataset. If there is more than one index object (each of a different type) for the same dataset, only one of them will generate a referral.

Though the format of the referral is dependent on the native protocol of the CIP server, the content of the referral is constant across all protocols. At the least, a DSI and a URI must be returned. The DSI is the DSI associated with the dataset which caused the hit. This must be presented to the client so that it can avoid referral loops. The other required piece is a URI. In general, this URI provides a compact way to return the hostname and port number that the client is being referred to. The URI was chosen for this field because it can hold additional information as necessary too. When an index object container is received with a Base-URI attribute in it, referrals based on that index will use that URI, instead of the default, which is generate from the hostname and port associated with the index object.

The additional information in the Base-URI may be necessary for the server receiving the referred query to correctly handle it. An good example of this is an LDAP server, which needs a base X.500 distinguished name from which to search. When an LDAP server sends a CENTROID up to a CIP indexing server, it sends a Base-URI along with the name of the X.500 subtree for which the index was made. When a referral is made, the Base-URI is passed back to the client so that it can pass it to the original LDAP server.

As usual, in addition to sending the DSI, a DSI-Description attribute can optionally sent. Because a client may attempt to check with the user before chasing the referral, and because this string is the friendliest CIP has to offer, it should be included in referrals when possible. Of course, the DSI-Description is only available for inclusion in referrals if it was sent the the index server as part of the index object.

* Cross-protocol Mappings

Each data access protocol which uses CIP will need a clearly defined set of rules to map queries in the native protocol to searches against an index object. These rules will vary according to the data domain. In principle, this could create a bit of a scaling difficulty; for N protocols and M data domains, there would be N x M mappings required. In practice, this should not be the case, since some access protocols will be wholly unsuited to some data domains. Consider for example, a LDAP server trying to make a search in an index object composed from Web pages. What would the results be? How would you even make sense of the incoming query or the outgoing results?

However, as pre-existing protocols are connected to CIP, and as new ones are developed to work with CIP, this issue must be examined. In the case of Whois++ and the CENTROID index type, there is an extremely close mapping, since the two were designed together. When hooking LDAP to the CENTROID index type, it will be necessary to map the attribute names to attribute names which are already being used in the CENTROID mesh. It will also be necessary to tokenize the LDAP queries under the same rules as the CENTROID indexing policy, so that searches will take place correctly.

* Moving through the mesh

From a client's point of view, CIP simply pushes all the "hard work" onto its shoulders. Afterall, it's the client which needs to track down the real data. While this is true, it's very misleading. Because the client has control over the query routing process, the client has total control over the size of the result set, the speed with which the query progresses, and the depth of the search.

The simplest client implementation simply provides referrals to the user in a raw, ready-to-reuse form, without attempting to follow them. For instance, one Whois++ client, which interacts with the user via a Web-based form, simply makes referrals into links. Encoded in the link via the HTML forms interface GET encoding rules is the data of the referral: the hostname, port, and query. If a user chooses to click on the referral, they execute a new search on the new host. A more savvy client might present the referrals to the user and ask which should be followed. And, assuming appropriate limits were placed on search time, bandwidth usage, etc, it might be reasonable to program a client to follow all referral automatically.

However, when following all referrals, a client must show a bit of intelligence. Remember that the mesh is defined as an interconnected graph of CIP servers. This graph may have cycles, which could cause an infinite loop of referrals, wasting the servers' time and the client's too. When faced with the job of tacking down all referrals, a client must use some form of a mesh traversal algorithm. Such an algorithm has been documented for use with Whois++ in RFC-1914. The same algorithm can be easily used with this version of CIP. In Whois++ the equivalent of a DSI is called a handle. With this substitution, the Whois++ mesh traversal algorithm works unchanged with CIP.

Finally, the mesh entry point (i.e. the first server queried) can have an impact on eh success of the query. To avoid scaling issues, it is not acceptable to use a single "root" node, and force all machine to connect to it. Instead, clients should connect to a reasonably well connected (with respect to the CIP mesh, not the Internet infrastructure) server. If no match can be made from this entry point, the client can expand the search by asking the original server who polls it. In general, those servers will have a better "vantage point" on the mesh, and will turn up answers that the initial search didn't. The mechanism for dynamically determining the mesh structure like this exists, but it not documented here for brevity. See RFC-1913 for more information on the POLLED-BY and POLLED-FOR commands.

CIP in the real world

Much of what we have discussed here appears to simply form a framework for future work. While it is true that there are many productive directions to go with CIP from here, CIP is already in use in at least one application (Whois++), and it is enlightening to explore another imminent application (Web indexing).

* Whois++ and CIP

CIP evolved out of work originally done on the Whois++ directory service system. Though it was originally designed for directory service tasks, the Whois++ indexing service was abstracted into what is now being developed as CIP. Thus, and early version of CIP is alive and in real use today in the form of Whois++ clients and servers.

Whois++ deals with records that are called templates. A template is an ordered list of attribute/value pairs. Multiple pairs with the same attribute are allowed in a given template. Typical search requests on a directory service revolve around finding the template with a certain word, or set of words in it. For instance, the search for "Jeff and Allen" matches all templates with both of those words in any attribute. In addition to unconstrained searches, it's also useful to constrain a keyword to a single attribute name, i.e. "Jeff and Allen and company=Bunyip".

To support this kind of searching in a distributed context, Whois++ servers exchange centroids. To create a centroid, first the records are broken up into tokens. The data is broken on spaces and other punctuation characters to form these tokens. The centroid is a list of unique tokens in the database sorted and separated by template name and attribute name. The centroid is case sensitive.

The design of the centroid was predicated by the searching and privacy requirements. It was necessary to allow keyword searching in a set of attributes and values. It was necessary to constrain keywords to particular attributes and/or values. It was required that the centroid not divulge the original database (thus the requirement for collation and duplicate removal). Though the terms had not been coined yet, the centroid designers were making the first index object by defining the first set of indexing policies. All that was left was to define the transfer format, and the distributed system was ready to go to work.

* CIP and the Web

Whois++ is a "textbook" case of applying CIP, since the two were designed hand-in-hand. What happens, one may ask, when CIP meets a new problem? Below, we discuss at the same level the design of a hypothetical index object called the weboid, used to index World Wide Web (WWW) content.

We start by exploring the requirements for the data domain to see if it is like any other data domains which are already being successfully indexed by CIP. If we can use one of those sets of indexing policies, our work will be done, and there will be no need for the weboid. First, web searching requires keyword searches. Next, metadata in the form of attribute value paris must be searchable (i.e. author, title, filetype). So far, it looks like we are simply talking about reinventing the centroid here. The final two requirements are what tell us we need a new indexing object. We need to be able to do searches on adjacency (i.e. "Jeff near Allen"), and we don't care too much how close the index is to the actual data, though we'd like to see some compression from the indexing process, or else we'll be faced with a system as unworkable as the systems in use today.

There are some other observations researches in the web indexing field have made that come to bear on our problem. First, very unique words (i.e. misspellings, acronyms, and exceedingly long words) will not be useful in the finished database. If a word is unique enough to occur only once in 1 million web pages, how likely is it to be used by a user during a search? Also, various stemming algorithms can greatly reduce the size of the index while increasing the yield of queries, making both administrators and users happy.

Because of the last two requirements, and because of the special characteristics of the web indexing data domain, we choose to make a

new indexing type. The weboid is born! The actual design of the weboid is left as an exercise to the reader.

Future Directions

CIP is a work in progress. It developed out of the indexing component of Whois++ [RFC-1913]. It became clear the centroid-passing ideas originally used in Whois++ would be useful in other realms, if it could be abstracted out of the Whois++ design. Though much progress has been made toward this end, work still needs to be done.

* Multi-domain index management issues

Research needs to be done into the topic of managing the interface of multiple CIP data domains. First, the question must be answered, "Is it useful to attempt to merge index objects from multiple domains?" Because data that is useful in many ways and via many different search techniques is likely to be indexed with CIP, it would seem important to make full use of the CIP meshes, instead of forcing them to remain forever isolated from one another.

However, even merging centroids with differing tokenization algorithms appears quite challenging. It will be necessary to come to grips with what it means to merge index objects from multiple domains, and to understand under what conditions this can be reasonably accomplished.

* Future Data Domains

As Internet contents becomes more varied, the demands on search and retrieval software do too. Because of the wide variation in data that is proliferating on the Net, CIP is designed to be extensible. Of course, this extensibility is only useful if we take the next step and actually develop alternative indexing objects for CIP.

Work is progressing on common indexing formats for web documents [IIF, DISW]. By applying CIP to web indexing using the example above (see "CIP and the Web") as a starting point, a scalable Web indexing system should soon be within our grasp.

Other challenging and exciting applications domains await, including indexing pictures and sounds.

* Mesh Management

Currently, CIP servers perform their polling duties with respect to a static configuration programmed by server administrators. There is no support in the protocol to change these polling relationships, thus all such configuration happens locally implementation-specific programs. In the future, we;d like to see mesh management take place over the TCP connection itself, so that configuration can be

accomplished remotely. Of course, this will require advances in the authentication methods used, as well as a protocol for the management itself.

In the distant future, it's reasonable to imagine groups of CIP server "managing themselves". A group of CIP servers which have been given the tools to measure the optimality of the mesh, have been given goals to reach for, and an algorithm to get them to the goal may be able to dynamically reconfigure a mesh to minimize the amount of polling overhead while maximizing client response time (by reducing the number of referrals in the system).

* Domain Identification

Because CIP is now extensible, there needs to be a way for CIP peer servers to enquire what data domains a particular server supports. This should be a simple addition to the protocol.

Security Considerations

There are two distinct levels at which security must be discussed with respect to CIP. First, we must explore the security necessary when dealing with indexing data. There are also standard issues to be explored with respect to protocol security.

* Secure Indexing

CIP is designed to index all kinds of data. Some of this data might be considered valuable, proprietary, or even highly sensitive by the data maintainer. Take, for example, a human resources database. Certain public bits of data, in moderation, can be very helpful for a company to make public. However, the database in its entirety is a very valuable asset, which the company must protect. Much experience has been gained in the directory service community over the years as to how best to walk this fine line between completely revealing the database and making useful pieces of it available.

Another example where security becomes a problem is for a data publisher who'd like to participate in a CIP mesh. The data that publisher creates and manages is the prime asset of the company. There is a financial incentive to participate in a CIP mesh, since exporting indices of the data will make it more likely that people will search your database. (Making profit off of the search activity is left as an exercise to the entrepreneur.) Once again, the index must be designed carefully to protect the database while providing a useful synopsis of the data.

One of the basic premises of CIP is that data providers will be willing to provide indices of their data to peer indexing servers. Unless they are carefully constructed, these indices could

constitute a threat to the security of the database. Thus, security of the data must be a prime consideration when developing a new index object type. The risk of reverse engineering a database based only on the index exported from it must be kept to a level consistent with the value of the data and the need for fine-grained indexing.

* Protocol Security

During the automated exchange of indexing information, there must be provisions made for CIP servers to identify themselves to one another. Though it's conceivable that a data server would be willing to make it's index available on an anonymous basis, most index objects will be passed between servers participating in a well-defined, statically configured indexing relationship.

In the existing implementations of CIP, only clear-text password authentication is available. When a poll request is submitted, a hostname and password are sent along. If the receiving server does not recognize the poller, the index object will not be sent. Likewise, when a "DATA-CHANGED" command is sent, it is ignored by the receiving server unless the sender's authentication information is valid.

Clear-text passwords are clearly not an acceptable authentication mechanism for use in the long term on the Internet. CIP either needs to rely on lower level authentication (like that provided by parts of IPv6) or it must be wrapped in a secure communication layer, such as SSL. Due to the complexity of the cryptography field, especially with respect to commercial interests, solutions to the authentication weakness in CIP have not been attempted yet.

Acknowledgements

Generous thanks to Leslie Daigle, Erik Selberg, and Roland Hedberg for comments on previous drafts of this paper.

References

[DISW] Erik Selberg, "DISW Query Routing Breakout Notes", June 1996. http://www.cs.washington.edu/homes/speed/disw-wu.html

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

[RFC-1914] P. Faltstrom, R. Schoultz, C. Weider, "How to Interact with a Whois++ Mesh", February, 1996

[INDEX500] David Chadwick. "IndeX.500", May 1996. http://www.dante.net/pubs/dip/19/19.html

[CENTIPEDE] T. Howes. "SLAPD and SLURPD administrator's guide".

http://www.umich.edu/~rsug/ldap/doc/guides/slapd

[IIF] Kevin Chang, Hector Garcia-Molina, Luis Gravano, Andreas Paepcke. "Internet Information Finding" http://www-db.stanford.edu/~gravano/standards

[IANA] Internet Assigned Number Authority http://www.isi.edu/iana

Author contact info

Jeff R. Allen
Bunyip Information Systems
Suite 300
310 Ste-Catherine St. West
Montreal, Quebec H2X 2A1
Canada

Phone: +1 514 875-8611
Fax: +1 514 875-8134
EMail: jeff@bunyip.com

Patrik Faltstrom
Tele2/Swipnet
BOX 62
S-164 94 Kista
Sweden

Phone: +46-8-56264000
Fax: +46-8-56264200
Email: paf@swip.net