The CCSO Nameserver – Why?

by
Steven Dorner s–dorner@uiuc.edu
Computer and Communications Services Office
University of Illinois at Urbana

December 7, 1989

updated by
Paul Pomes paul–pomes@uiuc.edu
Computer and Communications Services Office
University of Illinois at Urbana

August 2, 1992

Introduction

There are several documents that describe the function, implementation, and use of the CCSO Nameserver. This paper has a slightly different thrust; it endeavors to answer the question, "Why?" Why did we want a Nameserver? Why did we put in the features we did, and not others? Why did we develop our own rather than use someone else's? Why did we choose the CSnet server as a starting point, and what are the differences between our Nameserver and the CSnet server?

This paper should be of most use to those contemplating a name service for their own organization. We hope it will at least point such people at some of the questions to ask about a name service; we do not pretend that they will reach the same conclusions as we did.

Necessary to the understanding of this paper is *The CCSO Nameserver – A Description*. *A Description* explains exactly what our Nameserver is, and how it operates. We will assume that the reader is familiar with that information, and will not attempt here to duplicate the explanations offered in *A Description*.

The CCSO Nameserver is, like many systems, the product of design, accident, and evolution. Not everything panned out as we had hoped; some things we thought were important were eventually discarded; some things we tried did not succeed. Special note will be made where the reality differs radically from the intention. These places merit close scrutiny, since they indicate a place where the "right" choice is probably not very clear.

Why Have a Name Service?

The first question to be answered is, why do this thing at all? What are the real benefits of a name service like the CCSO Nameserver? We had several reasons for expending the effort to create our Nameserver.

- **Electronic mail directory.** First and foremost, we wanted some way for people in our University to find one another's electronic mail addresses. There are hundreds of computers in many different departments, and finding somebody's email address was like looking for a needle in a haystack. A name service would provide a central collection and inquiry point for electronic mail addresses. We have found

Converted to portable n/troff format using the -me macros from funky Next WriteNow format (icch).

the Nameserver to be very useful in this regard, although it has been difficult to collect the electronic mail addresses.

- **Automatic mail forwarding.** Another reason, ironically, that we wanted a Nameserver was to eliminate the need for explicit electronic mail addresses. A central repository of specific email addresses would give us the capability to accept very general addresses, such as “Steve.Dorner@uiuc.edu”, turn them into proper electronic mail addresses, and deliver them. This would not only eliminate the need for correspondents to know email addresses, but would also allow people to move from machine to machine without having to retrain everyone who sends them mail. It would only be necessary to change the electronic mail address in the name service to change the account that receives a person’s mail. In practice, we have found this to be a very convenient service.
- **Electronic “Phone Book”.** Another function of a name service would be as a replacement for the paper phone book. It could be used to look up mailing addresses and phone numbers of people or campus organizations, just like a regular phone book. Unlike a regular phone book, it would be ready at the fingertips of anyone on our campus network, not across the room on a bookshelf. Also unlike a regular phone book, it could be kept up to date as people move around in the University. We are a little suprised that this is **the** big hit of our Nameserver, and the major reason that people use it.
- **User validation.** Given a name service that keeps some information about everyone on campus, it is reasonable to contemplate using it as a central point for authentication. A user would identify himself to the name server, and other systems would accept the name server’s word that the user was who he said he was, eliminating the burden on the user of remembering many different passwords, as well as ensuring security. This was not to be part of our initial implementation, however. *[Reality: This has been put on hold until MIT’s Kerberos ceases to be a moving target.]*
- **Accounting.** The name service could serve as a central repository for accounting information.

What is has come down to for us is: the Nameserver is a very good substitute for the paper phone book, a substitute that people really like; the Nameserver is the only way to find out someone’s email address; and the Nameserver is very useful for forwarding mail.

What Did We Want In a Nameserver, and Why Did We Want It?

We had definite ideas about some of the features we wanted in our name service. The following items were considered essential:

- **Flexibility.** We wanted our name service to be a fairly general tool. Rather than try to think of all possible uses of it beforehand, our goal was to come up with a design that would give us the freedom to add new data items or new categories of entries. The keeping of tagged fields and the use of field properties in our Nameserver have met this goal completely.
- **Large Capacity.** Obviously, we needed a database that could handle the fifty to one hundred thousand entries we were expecting. This is actually a rather magic range; in the middle of the range, one exceeds the capacity of 16 bit pointers. Our Nameserver uses 32 bit pointers, and so is quite safe from a size standpoint.
- **High Performance.** The system has to be very fast if it is going to be used; no one is going to want to wait a long time to look up a phone number. Moreover, low delay is absolutely essential if a name service is going to be involved in mail handling. Response time for our Nameserver on a typical query is 1 second, and most of that is spent on network setup, so we are pleased with performance.
- **Online Update by Users.** We wanted a name service in which individuals could update their own information. There were basically three reasons for this; two practical and one philosophical. The practical reasons are that we didn’t want to bear the large clerical burden of keeping the database up-to-date, and we didn’t want to incur the inevitable time and accuracy penalty that goes with the aforesaid clerical burden. The philosophical reason is we’d rather users have control of their own information, unless there was a good reason for them not to do so. We allow anyone to change most information about themselves in our Nameserver; they can do it any time they wish, and the changes are instantly available to others.
- **Network-based.** Obviously, a large database that is being dynamically updated is going to have to be accessed over a network. Even if the database wasn’t too large to be replicated on each computer, the

fact that updates are possible at any time would mean the databases would be out of date immediately. A distributed system such as that used by the Internet Domain Name Server[1] could have been used; both implementation restraints and security concerns made us decide in favor of a single, centralized database.

We feel that our Nameserver has met all of the essential requirements listed above. We also had a list of items that would be nice, but were not considered essential, for a name service. Our track record with these items is less good; some of them were not done because we changed our minds about their worth, others because we just didn't have the time to implement them.

- **The Owning Account.** This idea came from the CSnet Name Server;[2] the gist of it is that each user has a single account which "own" his or her entry. The name service only requires that the host on which the account resides verify that the account accessing the entry is indeed what it is claimed to be; no passwords are required of the user. *[Reality: We never did this. It requires that the hosts on which owning accounts reside have a high degree of cooperation with the name service, something we did not feel we could get from all the University computers. We could do this for those we administer ourselves, but have not felt a great need to do so. Instead, we assign passwords to anyone who wishes to change his or her information.]*
- **Domain-based Authorization.** Along with the concept of the owning account came the idea of domain-based authorization. In short, one viewed the owning account as being composed of several domains, each of which would have its own Nameserver entry. Further, each of these domains would be allowed to do updates on the information kept about anyone whose owning account was in its domain. For example, if the owning account for my entry was "dorner@garcon.cso.uiuc.edu", there would be five "people" who could update my entry:

dorner@garcon.cso.uiuc.edu	(me)
garcon.cso.uiuc.edu	(my system administrator)
cso.uiuc.edu	(my department)
uiuc.edu	(my University)
edu	(the super-user)

This would allow us to restrict some fields in our name service to being changed at certain levels; for example, only my department (or above) could change my job title. This would maintain the integrity of the database (I couldn't lie about myself), and at the same time would not place the burden of upkeep on any single area; each domain would handle its own. *[Reality: We never implemented this scheme. Most data elements are changeable by the individuals involved; if they wish to lie, they may do so. About the only thing that is restricted is the name of the person; changes to names are handled by me personally at this time. I had to make five changes in a year of operation.]*

- **Wide Availability of Client Software.** To quote from one of our early design documents:

  > People should be able to access the nameserver from just about anything short of an Edsel.

  We wanted a name service that was available to anyone on our campus network. In this we have done fairly well; we have clients for UNIX, VMS,[3] DOS,[4] Macintosh,[5] as well as a limited VM/CMS client.
- **Easy Entry of Information.** With at least 50,000 entries expected, we had to have an automated way of entering information; "just type it in" was right out. In this our NameServer gets mixed reviews.

[1] See RFC-1035, *Domain Names – Implementation and Specification*, P. Mockapetris.

[2] See *The CSNet Name Server*, M. Solomon, L. Landweber, D. Neuhengen.

[3] The VMS client was contributed by Mark Sandrock (m-sandrock@uiuc.edu) of the University's School of Chemical Sciences.

[4] The DOS client was contributed by Gary Jacobs of Qualcomm Corp (gjacobs@qualcomm.com).

[5] The Macintosh client was contributed by John Norstad, Academic Computing and Network Services, Northwestern University (j-norstad@nwu.edu).

Automated entry is used, but it is not very pleasant; witness the document, *Rebuilding a Nameserver Database In 24 Easy Steps*.

- **Don't Pass Passwords Over the Network.** With cheap ethernet devices available, it is fairly easy for the asocial person to tap an ethernet and grab packets. This would allow such a miscreant to steal passwords; to avoid this, we wanted no password to traverse our networks "in the clear". Our Nameserver uses a random challenge scheme that meets this requirement.

Why Did We Develop Our Own?

Some organizations suffer from NIHS (Not Invented Here Syndrome); if someone else did it, it's not good enough for us. We'd like to think that does not describe us. While we were thinking about criteria for a name service, we also surveyed the field to see if any of the name servers then in existence were appropriate for out task. We looked initially at two; the CSnet Name Server and White Pages from Andrew;[6] later, we examined NetDB from Stanford.[7] In tabular form, this is what we found:[8]

Criteria	CSNET	White Pages	NetDB
Flexible?	No	No	No
Large Capacity?	No; 16 bit pointers.	No; performance problems with only a few thousand entries.	Yes
Performance?	Yes	Problems with a few thousand entries	Not evaluated
Online Updates?	Yes	No	Clerical staff only
Network-based?	Yes	Yes	Yes
Owning Account?	Yes	N/A	N/A
Domain Auth?	No	N/A	Sort of
Widely Avail Client?	Yes	No; need Andrew	Yes
Easy Info Entry?	Yes	Unknown	Yes
Secure Passwords?	Yes	N/A	N/A

Table 1. Name Server Comparison

The above comparison made it clear to us that we couldn't use any of the three candidates as they were. They simply didn't have the features we felt were essential. Therefore, we decided to roll our own.

Why Did We Use the CSnet Server As a Starting Point?

We did not start completely from scratch. We decided that the CSnet Name Server would make a good starting point for our own name server. Take a look at Table 1 again. The CSnet server met three out of five of our essential criteria, and four out of five of our list of non-essentials. It fell down in three areas: flexibility, by keeping a fixed set of information about each entry; capacity, by using 16 bit pointers; and the domain authorization scheme, which it did not use.

Examination of the CSnet source code revealed that the two major deficiencies, the 16 bit pointers and the inflexibility, could be remedied very easily. The underlying database was actually fairly general, and adapted itself easily to tagged fields and 32 bit pointers. That left only the domain authorization scheme,

[6] See *An Overview of the Andrew Message System*, J. Rosenberg, C. Everhart, N. Borenstein.

[7] See *User's Guide for NetDB*, Networking and Communications Systems, Stanford University.

[8] The information in this table was taken from before cited documents describing the three name servers, as well as from other documents and source code in the case of the CSnet server.

which we decided we could add easily enough at a later date. *[Reality: As was mentioned before, we never did do this.]*

What's the Difference?

In the process of adapting the CSnet Name Server to our purposes, we wound up making many changes. In fact, while we still use a modified version of the CSnet server's database code, the overlying software is all new. For the benefit of those familiar with the CSnet server, let us outline the differences between the CSnet software and our own:

- **Pointers expanded to 32 bits.** 16 bits translates into 65,000 entries (or 32,000 for signed pointers), and was not enough. We therefore increased the pointer size.
- **Fixed fields replaced with tagged fields.** The CSnet server had a half dozen or so null-terminated ASCII fields. Each field had to be present in every entry, although a field could be empty. The database proper (as opposed to the higher levels of the server) **did** allow an arbitrary number of null-terminated fields; a little surgery was done to exploit this basic orientation. As mentioned elswhere, each entry in our database has only non-empty fields, tagged with what kind of field each is.
- **One-tier authorization.** The CSnet server thought in terms of user, host, site triples. A user could change his information. A host could change the information of any user whose entry was "owned" by an account on that host. A site could change the entry of any host identified with that site. Finally, there was a "super-user" entry. Further, each host and site had a password that was used for client-server communication. We removed this entire structure; in our Nameserver, each user has his own password, and can use it to change his own entry. Some users have "hero" privileges, which allow them to change anything in any entry.
- **Removal of pipe and email interfaces.** The CSnet server had a client that could talk to it via the network, a pipe, or electronic mail. We remove the latter two capabilities (although our server **can** be talked to through a pipe by server administrators). The pipe was a simple special case of network access, and was removed without loss of functionality (nor efficiency, since our server is on a machine with no real users). The email interface could be done, but we have had no incentive to do it; network access has so far been sufficient.[9]
- **Removal of mail forwarding programs.** The CSnet server looked at mail forwarding very differently than do we; we therefore removed that portion of the code. Mail forwarding is done outside of the Nameserver *per se*, in other programs. These programs use the Nameserver to look up information, but are otherwise unrelated to it.
- **New Clients.** Our client software is completely different. The ideas are basically the same, but the commands and other operational issues have all been addressed afresh.
- Our Nameserver is talked to very differently than the CSnet server. We found the CSnet protocol somewhat confusing, and rather difficult for a human to use; ours can actually be used by a *homo sapiens* without great grief.

There are of course many other differences. The list above is only intended to hit the "high spots", places where we differ in a significant manner from the CSnet product. Note that we have not necessarily *improved* on CSnet's work; we have *adapted* some of it to fit our needs and desires. It may well be that the CSnet server would be more appropriate for any given organization. For example, if network connections are not universal our server is a bit awkward; the built-in email interface of the CSnet server would be quite useful in such a case.

Conclusion

We are quite pleased with our Nameserver. It meets most of our needs, and we are confident that its flexibility and good performance will serve us well as we use it for more purposes. Organizations considering a

[9] There is actually one quasi-mail interface to our Nameserver; it runs on our VM/CMS machine and translates BITnet messages into Nameserver queries.

name service are welcome to evaluate ours, and use it if they wish (subject to the distribution conditions outlined in *The CCSO Nameserver – A Description*, which are liberal). There are however other excellent choices; your choice will depend on your needs.