The *Hesiod** Name Server

Stephen P. Dyer

Project Athena
Massachusetts Institute of Technology
Cambridge, MA 02139
dyer@ATHENA.MIT.EDU

ABSTRACT

Hesiod, the Athena name server, provides naming for services and data objects in a distributed network environment. More specifically, it replaces databases that heretofore have had to be duplicated on each workstation and timesharing machine (e.g., remote file system information, */etc/printcap*, */etc/services*, */etc/passwd*, */etc/group*) and provides a flexible mechanism to supply new information as the need arises.

1. Introduction and Purpose

The computing enviroment at Project Athena has recently changed from a group of timesharing machines to a collection of file servers and many hundreds (and potentially many thousands) of publically-accessible workstations. The origins of UNIX† as a time-sharing system become acutely obvious when confronted with the need to manage information for hundreds of machines that may be used by many different individuals. The method used by UNIX to maintain information for its users and programs has been ASCII database files stored on each machine which are authoritative for all users of that machine. However, this breaks down when the number of machines and potential users are multiplied by two or three orders of magnitude. The system management effort to keep each machine's information current grows directly as the number of machines; this quickly becomes unworkable with more than a few dozen machines. We wanted a solution that could easily accomodate Athena's expected growth for the next 5 years.

Rather than having information duplicated on each machine, the concept of retrieving information via a network service, a *name server*, has proved workable and reliable. Xerox's *Clearinghouse*,[1] Sun's *Yellow Pages*[2] and the Internet Domain Name Server[3,4] are examples of name services in current use. We chose to base our name service, *Hesiod*, on the Berkeley Internet Domain Name Server, BIND,[5] for several reasons. First, the design had proved itself through its use in the Internet over the past several years, and it had a number of features that made it an attractive base for *Hesiod*: its hierarchical name space, the ability to delegate authority to subsidiary name servers, and the ability to take advantage of local caching of data to improve performance. Second, the BIND source code was readily available and provided a firm foundation for a more general name service; we did not have to spend time building low-level support facilities which it already provided. Finally, BIND source code is non-proprietary, which would facilitate our distribution of *Hesiod* to other interested sites.

Hesiod provides a name service for use by workstations and timesharing systems. It does not address the problems of centralized management and distribution of such information, which is provided by another service, the Athena Service Management System, or SMS.6 SMS maintains and distributes information managed by Athena Operations to each of the Athena *Hesiod* name servers. *Hesiod* may be used without SMS;

* n. 8th century B.C. Greek poet. The names of the Gods and the myths surrounding them are recorded in his poetry.
† UNIX is a trademark of Bell Laboratories.

neither is dependent on the other. However, without an information management system front end, the *Hesiod* databases are simply ASCII files in BIND-compatible resource records format that must then be managed with a text editor. Large sites may appreciate the convenience SMS provides, while smaller sites may opt for the simplicity of using *Hesiod* without SMS.

Hesiod provides a content-addressible memory where certain strings can be mapped to others depending on the query. *Hesiod* has no knowledge about the data it stores; queries and responses are simple key/content interactions. It is designed to be used in situations where a small amount of data that changes infrequently needs to be retrieved quickly, with little overhead. It is not intended to serve as a general-purpose database system supporting arbitrary queries, or as a repository for information that changes frequently. The current implementation provides no facility for an arbitrary application to update the *Hesiod* database, which is refreshed several times a day by the Athena SMS. Because of the limitation imposed by the underlying implementation of *Hesiod*, based as it is on the Internet domain naming scheme, there is a maximum length of 512 bytes of data that can be exchanged between the client and the name servers using UDP datagrams. This imposes limits on both the maximum size of an individual data record, as well as the number of records that can be returned in a single packet in the case of multiple matches. *Hesiod* was designed to provide applications with a rapid, low-overhead naming service in which a query would return no more than a few matches of limited size. Applications that require more complicated queries or ones that return voluminous data should consider interfacing to SMS.

2. *Hesiod* Queries

A *Hesiod* query consists of two parts, a *HesiodName*, which is the name of an object in the network, and a *HesiodNameType*, an application-specific qualifier that identifies the application space in which that object is named.

We do not use standard Internet Domain Name notation to refer to *HesiodNames* for several reasons: First, we wish to have objects with name containing the '.' character.† In Internet domain notation, a name that contains a '.' is considered fully resolved. Second, early BIND implementations had no provision for deciding the proper domain suffix to use when resolving a relative name.

† As just one example, MIT course names, such as *6.001*, contain periods.

A name given to the *Hesiod* name server for resolution looks like:

```
HESIODNAME => LHS
HESIODNAME => LHS@RHS
LHS     => [Any ASCII character, except NUL and '@']*
             { 0 or more characters from this alphabet }
RHS     => [Any ASCII character, except NUL and '@']+
             { 1 or more characters from this alphabet }
```

In other words, a *HesiodName* consists of [LHS][@RHS] where either [LHS] or [@RHS] need not be present.

The LHS of a *HesiodName* is *uninterpreted*; although it may be modified according to the rules described by the information in */etc/hesiod.conf* (see below), it is not itself a domain name.

We define a set of routines known as the *Hesiod* library that take two strings, a *HesiodName* and a user-supplied key, a *HesiodNameType*, convert it to a fully-qualified domain name, call the BIND library, and return the results to the original caller. The *HesiodNameType* is a well-known string that is provided by an application that uses the *Hesiod* library. It is used directly in the expansion of a *Hesiod* name to a BIND name (see below) without further indirection or translation. A new *HesiodNameType* comes into existence simply by being used by an application; no libraries or configuration files need to be modified. Naturally, there has to be appropriate data stored by the name server which is associated with that *HesiodNameType*.

To provide an example, one of the routines in the *Hesiod* library takes a *HesiodName* and returns a fully-qualified name to be handed to BIND:

```
char *
hes_to_bind(HesiodName, HesiodNameType)
char *HesiodName, *HesiodNameType;
```

The *HesiodNameType* identifies the query to make to BIND and the proper expansion rules to use with the LHS and RHS of the name. This would be chosen by the application, and could be application-specific.

Thus, the following are valid *HesiodNames*:

14.21
default-printer

default-printer@SIPB
@heracles
@heracles.MIT.EDU
kerberos@Berkeley.MIT.EDU

The configuration file */etc/hesiod.conf* contains two tables specifying the treatment of LHS and RHS components of a *HesiodName*. In the translation of a *HesiodName* to a valid BIND name, the LHS is expanded by concatenating together the *HesiodName*, the separator '.', the HesiodNameType, and the LHS entry found in the configuration files. If the RHS is null, the RHS entry in the configuration file is used. If the RHS is a fully qualified domain name already, it is used directly. Otherwise, if a RHS is present, it is used as a *HesiodName* for further resolution against the *HesiodNameType*, "rhs-extension". If this query succeeds, the first reply is used as the RHS, otherwise an error is returned. The fully-expanded LHS and RHS are then concatenated together, separated by a '.', and this value is passed to BIND for resolution.

The following is a typical copy of */etc/hesiod.conf*:

```
#file /etc/hesiod.conf
#comment lines begin with a '#' in column 1
#LHS table
lhs = .ns
#RHS table
rhs = .Athena.MIT.EDU
```

With this definition, a call to **hes_to_bind**("e40", "printer") would produce a LHS of "e40.printer.ns" and a RHS of ".athena.MIT.EDU", and the resulting BIND name, "e40.printer.ns.Athena.MIT.EDU".

In C pseudo-code, we would have the following productions:

hes_to_bind("14.21, "filesys") =>
"14.21.filesys.ns.Athena.MIT.EDU"
hes_to_bind("e40, "printer") =>
"e40.printer.ns.Athena.MIT.EDU"
hes_to_bind("SIPB, "rhs-extension") =>
"SIPB.rhs-extension.ns.Athena.MIT.EDU"
hes_to_bind("default@SIPB", "printer") =>
"default.printer.ns.SIPB.MIT.EDU"
(this assumes that the previous production resolved to "SIPB.MIT.EDU")
hes_to_bind("kerberos@Berkeley.EDU", "sloc")
=> "kerberos.sloc.ns.Berkeley.EDU"

These productions are then passed to the BIND name server for resolution.

3. Data Types

Hesiod data are stored as Internet domain resource records. A new class, HS, signifying a *Hesiod* query or datum has been reserved, and a new query type, TXT, that allows the storage of arbitrary ASCII strings. Paul Mockapetris, the Internet Domain System designer, has recently specified the HS class and the TXT type in RFCs 1034 and 1035.

4. BIND Requirements

A version of BIND that supports the HS query class and TXT query type is required to support the *Hesiod* name service. The latest release of BIND as of 12/31/1987, version 4.7.3, has been modified at Athena to support this, and we will be forwarding these changes to Berkeley for future releases of BIND.

5. Athena Client Applications of *Hesiod*

Many applications and subroutines have been modified to take advantage of the *Hesiod* service. See Appendix A for an enumeration of some of the *HesiodNameTypes* in common use within Project Athena.

The *attach* program queries *Hesiod* for the filesystem with the given name, retrieves the data, and mounts the appropriate RVD or NFS[7] filesystems, while also authenticating the user to the file server using *Kerberos*.[8]

Login uses the user's login name as a *HesiodName* to retrieve the user's */etc/passwd* and group membership information. The actual password field is not used; rather the *Kerberos* service authenticates the user. *Login* queries *Hesiod* to determine which *Kerberos* server to invoke. By convention, the username is also the name of the user's default filesystem. The *login* program runs the *attach* program (*q.v.*) with the user's login name as an argument to mount the user's home directory.

Athena users receive their mail on POP[9] (post-office protocol) servers. We have modified the MH programs *inc* and *msgchk* to query *Hesiod* for the location of the user's POP server.

The *lpr* program is compiled with a special version of the */etc/printcap* access library that queries *Hesiod* if the printer name cannot be found in the local */etc/printcap*.

There are optional implementations of **getpwnam()**, **getgrnam()**, and their inverse counterparts that query *Hesiod* for name-to-UID, name-to-GID translation, and vice-versa. The same library includes an implementation of **getservent()** that queries *Hesiod* in preference to lookup in the file */etc/services*.

6. *Hesiod* Resource Records and Data Files

Appendix B lists samples of the resource records that we store on behalf of *Hesiod* client applications. The format of the ASCII strings returned by *Hesiod* is application-specific. In the case of queries that have an inverse operation, such as queries with the *HesiodNameTypes*, **passwd**, and **uid**, the **uid** resource records are *CNAMEs* for the corresponding **passwd** records.

The BIND boot file on each workstation, */etc/named.boot*, refers to an auxiliary cache file, */etc/named.hes*, that specifies the authoritative name servers for *Hesiod* queries.

7. Programming with the *Hesiod* Library

There are only two subroutines, **hes_resolve()** and **hes_error()**, that are usually invoked by the applications programmer when using *Hesiod*. The subroutine **hes_resolve()** is the primary interface into the *Hesiod* name server. It takes two string arguments, the name to be resolved, the *HesiodName*, and a type indicating the type of service associated with this name, the *HesiodNameType*. **hes_resolve()** returns a pointer to an array of strings, much like **argv[]**, containing all the data that matched the query, one match per array slot. The array is NULL terminated. A second call to **hes_resolve()** will overwrite any previously-returned data, so applications that require data to be maintained across multiple calls to **hes_resolve()** should copy the returned values into data areas they maintain.

Note that a call to **hes_resolve()** may return more than one match. The semantics of using or choosing between multiple matches is dependent on the particular application. In general, however, multiple matches are considered "equivalent", and any of them could be used equally well. This is exploited, for example, by the *attach* command that attaches a remote file system to the workstation. In the case of system libraries, multiple copies of which are considered equivalent, the attach command iterates through all matches, stopping after the first successful attach. Because *Hesiod* is based on the Internet Domain Naming scheme, no interpretation can or should be given to the order in which matches are returned.

If **hes_resolve()** returns NULL, then no data could be found, either because the name server had no matching records or an error occurred. The function **hes_error()** takes no arguments and returns a small integer indicating the type of error, if any, encountered in the last call to **hes_resolve()**.

It is important to emphasize that *Hesiod* knows nothing about the data it stores; any meaning given to the *HesiodName*, the *HesiodNameType* and the data returned by *Hesiod* is completely imposed by the application. The format of the data stored by *Hesiod* is application-specific, and would be defined by the application programmer.

```
#include <hesiod.h>

char *HesiodName, *HesiodNameType;
char **hp;

hp = hes_resolve(HesiodName, HesiodNameType);
if (hp == NULL) {
    err = hes_error();
    switch(err) {
    .
    .
    .
    }
} else {
    /* do your thing with hp */
    while(*hp != NULL) process(*hp++);
}
```

The error values returned by **hes_error()** are one of the following:

```
#define HES_ER_UNINIT      -1
#define HES_ER_OK           0
#define HES_ER_NOTFOUND     1
#define HES_ER_CONFIG       2
#define HES_ER_NET          3
```

The most common values returned by **hes_error()** are HES_ER_OK, meaning no error, and HES_ER_NOTFOUND, meaning that the desired name was not found in the *Hesiod* data base. HES_ER_CONFIG indicates a problem with the optional per-machine *Hesiod* configuration file, */etc/hesiod.conf*. HES_ER_UNINIT will never be returned by **hes_error()**, unless it is called before the first time **hes_resolve()** is called. HES_ER_NET indicates that the request never received a response from the *Hesiod* name server. This can be due to a variety of network problems: for example, the host making the request might be disconnected from the network, an intervening gateway might be down, or no *Hesiod* name servers responded. No further information about the state of the network is available because the domain system on which *Hesiod* is based uses datagrams with retries as the communications interface.

HES_ER_NOTFOUND is a negative acknowledgement indicating that the desired name/*HesiodNameType* pair was not found in the *Hesiod* database. An application receiving this error message can consider this an authoritative response. Of course, this may be due to an omission in the database, or simply reflect a delay between the time *Hesiod* data was asked to be placed into the database, and the actual *Hesiod* updates, which occur several times each day.

In the case of a *Hesiod* error of HES_ER_NET, it may be prudent for an application to assume that this situation is temporary, and that a later call to **hes_resolve()** will either return the desired data or a definitive reply of HES_ER_NOTFOUND. HES_ER_CONFIG indicates a problem with the *Hesiod* configuration file, a situation that requires intervention by a wizard and will not resolve itself spontaneously. Because no query to the *Hesiod* name server is actually made, no conclusion can be drawn about the validity of the name to be resolved. The standard Athena distribution of the *Hesiod* library does not require a configuration file; its built-in defaults suffice, so this situation should not be encountered frequently.

A general design strategy for applications using *Hesiod* is to have a contingency plan in place in case *Hesiod* does not respond, is configured incorrectly or does not know the name. This may be built-in to the application, such as new versions of *lpr* that revert to using the old printcap libraries if *Hesiod* printer information is not available. Another popular scheme, exploited by the *MH* application *inc* and the EMACS tool, *movemail*, is to allow the value of an environment variable, in this case, **MAILHOST**, to override the call to *Hesiod* to retrieve a person’s mailhost, using his username as the key. Thus, a user can temporarily "hardwire" appropriate values to allow applications to proceed. Not every application can be programmed in such a fashion, but it is prudent to try to design applications with this in mind.

8. Database Size and Performance

A measure of how successful *Hesiod* has been in its deployment over the past six months is how infrequently problems have appeared. For the most part, applications make *Hesiod* queries and receive answers with millisecond delays. Today, the *Hesiod* database for Project Athena contains almost three megabytes of data: roughly 9500 */etc/passwd* entries, 10000 */etc/group* entries, 6500 file system entries and 8600 post office records. There are three primary *Hesiod* nameservers distributed across the campus network.

BIND has proven itself remarkably robust in accomodating such a large, monolithic database. One problem has been noticed: the time to load the primary nameservers (which are updated from the Athena SMS every six hours) has increased markedly as the size of our data has grown. At this point, it takes approximately 20 minutes to reload a primary nameserver running on a VAX 750 and each primary nameserver’s working set is approximately 10 megabytes. By staggering the times to reload each of the primary *Hesiod* servers, this has not proved to be a large operational problem. However, it does point out an area that should be examined for improved performance. Because the *HesiodNameType* component in the domain name passed to BIND identifies a potentially separate start of authority, the *Hesiod* database could be split across two or more primary nameservers, each authoritative for a subset of the full database. This would reduce the time to load each nameserver and the size of its working set.

9. Acknowledgements

Jerry Saltzer, Technical Director of Project Athena, provided a great deal of assistance and guidance in the design of the *Hesiod* name server. Thanks, too, go to Dan Geer and Jeff Schiller for their assistance during the design and deployment stages. Clifford Neuman, presently at the University of Washington, and Felix Hsu of Digital Equipment Corporation participated in the early design of the system.

Appendix A: *HesiodNameTypes* in Use at Athena

Here is a list of some of the presently-defined HesiodNameTypes, the type of information provided as a *HesiodName*, and the applications programs that use such queries.

HesiodName	*HesiodNameType*	Used By	Info Returned
workstation name	"cluster"	*getcluster*	workstation cluster information
filesystem name	"filsys"	*attach/detach*	RVD and NFS file system info
username	"pobox"	MH *inc/movemail*	location and type of mailbox
username	"passwd"	*toehold/login* **getpwent()**, *et. al.*	Athena-wide /etc/passwd entry
uid (ASCII)	"uid"	**getpwent()**, *et. al.*	Athena-wide UID to username mapping
group name	"group"	**getgrent()**, *et. al.*	Athena-wide /etc/group entry (no membership list)
group name	"grplist"	**getgrent()**, *et. al.*	Athena-wide group membership mapping
gid (ASCII)	"gid"	**getgrent()**, *et. al.*	Athena-wide GID to group name mapping
printer name	"pcap"	**pgetent()**	Athena-wide /etc/printcap entry
service name	"service"	**getservent()**	Athena-wide /etc/services entry
service name	"sloc"	*On-Line Consulting (OLC)* *Kerberos*	Host name to contact for this service (for those services that do not reside on every host)

Appendix B -- Sample Resource Records from Current *Hesiod* Database Files

```
# filsys.db
# format of data is
#   filesystem-type name-on-server server-hostname mount-mode mount-point
dyer.filsys     HS      TXT "NFS /mit/dyer eurydice w /mit/dyer"
dyfeigen.filsys HS      TXT "NFS /mit/lockers/dyfeigen zeus w /mit/dyfeigen"
dyim.filsys     HS      TXT "NFS /mit/lockers/dyim zeus w /mit/dyim"
bldg1-rtsys.filsys      HS      TXT "RVD rtsys oath r /srvd"
bldg1-rtsys.filsys      HS      TXT "RVD rtsys persephone r /srvd"

# gid.db
# format of data is
#   canonical name with this group id
481.gid HS      CNAME   10.01.group
483.gid HS      CNAME   10.01a.group
484.gid HS      CNAME   10.01b.group
639.gid HS      CNAME   10.01sa.group
640.gid HS      CNAME   10.01sb.group
638.gid HS      CNAME   10.01t.group

# group.db
# format of data is
#   /etc/group entry
10.01.group     HS      TXT 10.01:*:481:
10.01a.group    HS      TXT 10.01a:*:483:
10.01b.group    HS      TXT 10.01b:*:484:
10.01sa.group   HS      TXT 10.01sa:*:639:
10.01sb.group   HS      TXT 10.01sb:*:640:
10.01t.group    HS      TXT 10.01t:*:638:

# grplist.db
# format of data is
#   groupname1:gid1:groupname2:gid2:...
10.01.grplist   HS      TXT "10.01:481:10.01t:638"
10.01ta.grplist HS      TXT "10.01t:638"

# passwd.db
# format of data is
#   /etc/passwd entry
dyer.passwd     HS      TXT "dyer:*:17287:101:Steve Dyer,,,,:/mit/dyer:/bin/csh"

# pobox.db
# format of data is
#   post-office-type server-host-name mailbox-name
dyer.pobox      HS      TXT "POP E40-PO.MIT.EDU dyer"

# printcap.db
# format of data is
#   /etc/printcap entry
nil.pcap    HS      TXT "nil|LPS-40:rp=nil:rm=castor.mit.edu:sd=/usr/spool/printer/nil:"
```

```
# service.db
# format of data is
#   service-name protocol port-number
discard.service HS      TXT "discard tcp 9"
discard.service HS      TXT "discard udp 9"
nntp.service    HS      TXT "nntp tcp 119"

# sloc.db
# format of data is
#   host name where this service is offered
zephyr.sloc     HS      TXT ARILINN.MIT.EDU
zephyr.sloc     HS      TXT NESKAYA.MIT.EDU
zephyr.sloc     HS      TXT ORPHEUS.MIT.EDU
zephyr.sloc     HS      TXT PRIAM.MIT.EDU

# uid.db
# format of data is
#   canonical name with this user id
17287.uid       HS      CNAME   dyer.passwd
```

References

1. Xerox Corporation, *Clearinghouse Protocol*, Stamford, Connecticut, April, 1984.
2. Sun Microsystems, *Yellow Pages Protocol Specification*, Networking on the Sun Workstation, 1986.
3. P. Mockapetris, *RFC 1034 - Domain Names - Concepts and Facilities*, USC/Information Sciences Institute, November, 1987.
4. P. Mockapetris, *RFC 1035 - Domain Implementation and Specification*, USC/Information Sciences Institute, November, 1987.
5. J. M. Bloom and K. J. Dunlap, ‘‘A Distributed Name Server for the DARPA Internet,’’ in *Usenix Conference Proceedings*, pp. 172-181, Summer, 1986.
6. M. A. Rosenstein, D. E. Geer, and P. J. Levine, ‘‘The Athena Service Management System,’’ in *Usenix Conference Proceedings*, M.I.T. Project Athena, Winter, 1988.
7. Sun Microsystems, *NFS Protocol Specification and Services Manual*, 1987. Revision A
8. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, *Section E.2.1: Kerberos Authentication and Authorization System*, Project Athena Technical Plan, M.I.T. Project Athena, Cambridge, Massachusetts, December 21, 1987.
9. M. T. Rose, *Post Office Protocol (revised)*, University of Delaware, 1985. (MH internal)