CITI Technical Report 93–5

The Design and Implementation of an AFP/AFS Protocol Translator

Thomas J. Hacker
hacker @citi.umich.edu

ABSTRACT

This paper gives an overview of the design and implementation of the AFP/AFS protocol translator currently in use at the University of Michigan. The protocol translator is an implementation of the AppleTalk protocol suite on BSD UNIX and BSD UNIX derivatives. The translator exploits currently existing UNIX TCP/IP mechanisms (such as sockets), and provides a programming interface to the session and transport layers of AppleTalk.

The translator is designed to export AFS and UNIX local file system components as AFP volumes. This ability enables users to access files in the large AFS file system using the native Macintosh interface. Additionally, the translator software on the Macintosh provides Kerberos authentication to the AFS client (AFS Kerberos), reauthentication for expired tokens (AFS Log), and the advantages of the rich access control mechanisms provided by AFS.

August 4, 1993

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

The Design and Implementation of an AFP/AFS Protocol Translator

Thomas J. Hacker

August 4, 1993

This paper describes the design and implementation of the AFP/AFS protocol translator[1] currently in use at the University of Michigan. Part of the Institutional File System (IFS) project [1], the protocol translator is an implementation of the AppleTalk [2] protocol suite on BSD UNIX [3] and BSD UNIX derivatives. The translator exports AFS [4] and UNIX local file system components as AFP volumes, enabling IFS users to use the native Macintosh interface to access files in the large AFS file system. Additionally, the translator software on the Macintosh provides Kerberos authentication to the AFS client (AFS Kerberos), reauthentication for expired tokens (AFS Log), and the advantages of the rich access control mechanisms provided by AFS.

AppleShare, using the AppleTalk Filing Protocol (AFP), is the native Macintosh file-sharing mechanism. A Macintosh client running AppleShare makes AFP requests to a server. The server then translates these requests into appropriate AFS requests, receives and caches the file returned by the server, and returns the file to the requesting client using the AFP.

1. Motivation and Background

As components of the Institutional File System (IFS) at the University of Michigan, protocol translators enable clients running operating systems other than UNIX to access the IFS. Protocol translation is a vital component in achieving the vision of the IFS project to extend AFS (its file system base) to support multiple platforms and a potential 30,000 workstations on the University of Michigan campus.

Protocol translators convert file system requests from one file system protocol to another. More than half of the campus community use Macintosh computers. Without the AFP/AFS protocol translators, these users would become isolated from the campus-wide, distributed computing environment.

2. User Interface Issues

The protocol translator looks like an AFP server to a Macintosh client. No changes were made to the AFP protocol or to existing software running on the client. However, some additional software

1. For programming-level details on the AFP/AFS protocol translator (formerly called Netatalk), refer to [5].

was installed on the Macintosh client to provide secure authentication and to manage AFS access control lists.

Accessing an AppleShare volume exported by a translator is no different than accessing a normal AppleShare volume. Users open the Chooser dialog box and select the file server they wish to access. After selecting a file server, they are prompted for authentication information. After authentication, users are presented with a list of available volumes—at minimum, their home directories.

To define the list of mountable volumes that appear after authentication, users can create a configuration file in their AFS home directories that maps a volume name to the corresponding portion of the AFS file tree. Figure 1 shows a sample list.

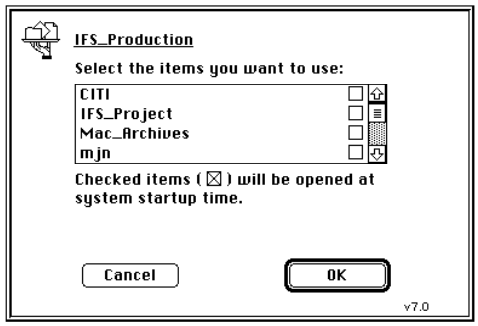


Figure 1. Sample Volume List

Selecting one or more volumes results in those volumes appearing on the desktop. For example, selecting the volumes CITI and Mac_Archives places the volume icons shown in Figure 2 on the desktop.

Figure 2. IFS volumes on the Macintosh desktop

Users manipulate these volumes as they would any other AppleShare volume. An AppleShare volume exported by an AFS server is simply a portion of the AFS tree.

2.1 Navigation

Macintosh users navigate the AFS file system as though it were a normal Macintosh volume. Double-clicking on an AFS volume icon opens it. Folders and files appear as the user expects. Users manipulate the directory hierarchy as they do regular Macintosh files.

2.2 Availability of AFS Commands

AFS provides several commands that allow users to view various characteristics of the file system. For example, commands exist that indicate the status of the available file servers.

For the Macintosh environment, however, only the authentication and ACL manipulation commands are currently available. (Most other commands have specific meaning only for AFS files.) Users do not use the AFS command itself, but instead use a program that fits the Macintosh paradigm and presents an equivalent capability. So instead of issuing a UNIX style command, users make choices and fill in values in a dialog box.

2.3 File Naming

File name compatibility presents a few problems. Most of the allowable characters in the Macintosh operating system are supported by AFS and vice versa. The notable exceptions are spaces, colons, and slashes. (The translator includes a mechanism that translates these characters into allowed UNIX file name characters.) Additionally, the Macintosh Finder imposes a maximum name length of 31 characters. File names in AFS that are longer than that appear truncated.

2.4 Authentication

The privacy of large scale file systems is important. Kerberos, developed by the Massachusetts Institute of Technology, provides a mechanism that keeps user authentication secure and avoids the transmission of passwords in clear text over campus data networks. However, the Macintosh client does not directly support Kerberos. In conjunction with the AFP/AFS protocol translator, authentication occurs through a new Macintosh program module that implements the Kerberos authentication mechanism through the Alternative UAM mechanism provided by the Chooser. Users select the Kerberos authentication method to access AFS file servers. After that, the user authentication dialog is identical to that for logging on to an AppleTalk server.

2.5 Permission Mapping (ACLs)

AFS uses access control lists (ACLs) that allow the user to specify permissions for AFS directories. Because AFP does not support identical permission modes, a new Desk Accessory (DA) is included with the translator. The AFS Privileges DA appears in the Apple desktop menu and enables users to set ACLs for folders (directories) contained in an AFS volume (Figure 3).

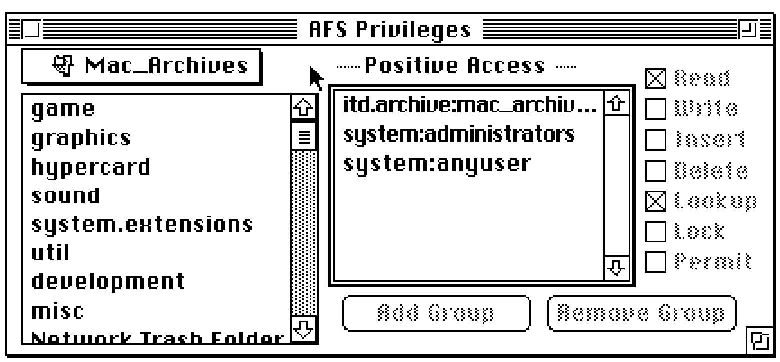


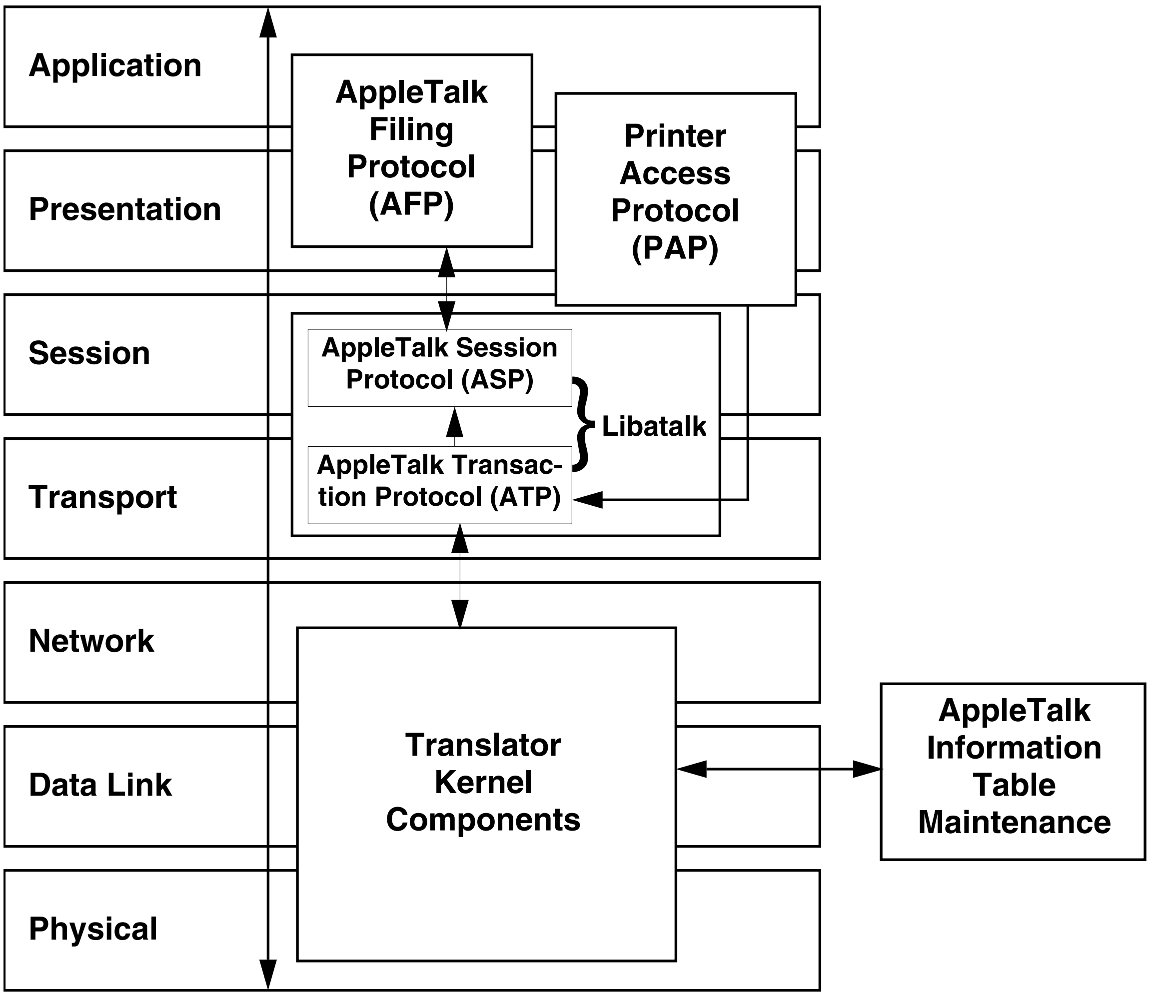
Figure 3. AFS Privileges Interface

3. Architecture

Underlying the user interface concerns, addressed above, is the internal architecture of the AFP/AFS protocol translator. The translator implements the AppleTalk protocol suite on BSD UNIX and its derivatives, exploiting currently existing UNIX TCP/IP mechanisms and abstract communication components such as bind, socket, sendto, and recvfrom. Some of the features of AFP/AFS protocol translator include the ability to ping AppleTalk hosts (aecho), to perform name lookups (NBP), a file service (AFP), and additional print services to the Macintosh (PAP).

Figure 4 shows the relationship of the various layers of the translator protocol stack and how they relate to the traditional ISO/OSI networking model.

Figure 4. Translator Protocol Stack



The AFP /AFS protocol translator is implemented primarily in the kernel of an intermediate server [6]. Besides the kernel level, the translator includes libraries (Libatalk) that provide a programming interface to the session and transport layers of its protocol. Libatalk provides the components necessary to implement such services as the AppleTalk Filing Protocol (AFP). The translator also supports native ports for implementing services such as AppleTalk aecho. Additionally, the translator provides network information table maintenance by supporting the Name Binding, Routing Table Maintenance, and Zone Information Protocols [5].

3.1 Translator Kernel Level

The AFP/AFS protocol translator includes kernel modules that process incoming and outgoing Datagram Delivery Protocol (DDP) packets and manage the AppleTalk Address Resolution Protocol (AARP) tables used for routing.

3.2 Information Table Maintenance

Outside the kernel level, the translator creates and maintains tables for the Name Binding Protocol (NBP), Routing Table Maintenance Protocol (RTMP) and Zone Information Protocol (ZIP) through a daemon process, `atalkd`. The translator provides several utilities to query and modify these tables. An additional application, `aecho`, allows a user to "ping" a host in a fashion similar to the TCP/IP ping.

3.2.1 NBP

The NBP portion of `atalkd` contains a linked list of structures describing the objects available from the AppleTalk stack on the host. Each structure describes an object, and a type. (For example, if "Poohbah" is an object and "LaserWriter" is a type, when an NBP query for =:= or =:LaserWriter is received, Poohbah:LaserWriter is returned.)

3.2.2 RTMP

The RTMP portion of `atalkd` receives RTMP information broadcast by routers, and manages a linked list of structures describing AppleTalk routes based on that information.

3.2.3 ZIP

The ZIP table maintained by `atalkd` is a linked list of structures describing the zone name. The mapping between zone name and network number is done by looking through the RTMP table.

3.2.4 AppleTalk Ports

The translator also supports the native AppleTalk ports: Routing Table Maintenance Protocol (RTMP), Name Binding Protocol (NBP), AppleTalk Echo Protocol (AEP), and Zone Information Protocol (ZIP). The complete list, formatted in a similar fashion as TCP/IP ports, appears as follows:

```
rtmp  1/ddp    Routing Table Maintenance Protocol
nbp   2/ddp    Name Binding Protocol
echo  4/ddp    AppleTalk Echo Protocol
zip   6/ddp    Zone Information Protocol
```

3.3 Libatalk: Translator Libraries

Above the kernel, Libatalk is a set of libraries that consists of the AppleTaIk Transaction Protocol (ATP), which handles packet management, and the AppleTalk Session Protocol (ASP), which maintains session information. Libatalk provides layers above the translator kernel level to support the higher-level applications AFP and the Printer Access Protocol (PAP). Programmers can use Libatalk to write additional applications at this level.

3.4 AFP Translator Implementation

AFP is Apple's remote file service. It allows mounting of remote volumes, and has mechanisms for access control and user authentication. The translator provides a platform for AFP on UNIX. AFP runs as a daemon process, and provides Kerberos authentication using the Alternative User Authentication Mechanism (UAM) on the Macintosh [7, 8], and translates AFP calls into file accesses to the UNIX file system.

3.5 PAP Translator Implementation

PAP, as it exists in the translator, provides print server service to Macintoshes accessible to the translator via AppleTalk. The print jobs it receives are queued up via the normal UNIX `lpq` mechanisms, and hence can be either printed locally or remotely through a TCP/IP network using the remote network printing capabilities of `lpq`.

4. Library for AppleTalk (Libatalk)

The Libatalk portion of the translator consists of two protocol layers that provide libraries and utility routines used in applications such as `afpd` and aecho. The first protocol layer and the lower of the two, ATP, supports a loss free delivery of client packets from a source socket to a destination socket. ATP does windowing for groups of packets, and ensures proper sequences. The second layer, ASP, uses the services of the ATP layer and is responsible for setting up and tearing down sessions, sending commands to a server, receiving command replies, writing blocks of data from the workstation to the server, and other services.

4.1 ATP

The AppleTalk Transaction Protocol (ATP) layer provides a loss free delivery of packets from a source socket to a destination socket. The translator kernel implementation of ATP implements exactly-once (XO) transactions, but not at-least-once transactions (ALO). The XO transactions guarantee non-duplicate requests, and the delivery of up to eight packets of data per request. ATP sequences packets (in most cases) and guarantees the packets are delivered in order.

The calls that Libatalk provides for using ATP are: `atp_open`, `atp_rreq`, `atp_rresp`, `atp_sreq`, and `atp_sresp`. The `atp_handle` (from include/`atalk/atp.h`) structure represents the open sockets in the ATP layer. The structure of `atp_handle` is:

```
struct atp_handle {
    int                 atph_socket;      /* ddp socket */
    struct sockaddr_at  atph_saddr;       /* address */
    u_short             atph_tid;         /* last tid used */
    u_short             atph_rtid;        /* last received (rreq) */
    u_char              atph_rxo;         /* XO flag from last rreq */
    struct atpbuf       *atph_sent;       /* packets we send (XO) */
    struct atpbuf       *atph_queue;      /* queue of pending packets */
    int                 atph_reqtries;    /* retry count for request */
    int                 atph_reqto;       /* retry timeout for request */
    u_char              atph_rbitmap;     /* bitmap for request */
    struct atpbuf       *atph_reqpkt;     /* last request packet */
    struct atpbuf       *atph_resppkt[8]; /* response to request */

/* the following for dynamic adaptive reqto */
#define NTRELS 4
    struct timeval      atph_treldly[NTRELS];/* delays for last N trel's */
    u_char              atph_trels;       /* next trel to use */
};

typedef struct atp_handle *ATP;
```

Another structure is the ATP parameter block, used in the ATP calls to get data in and out of the ATP layer. The structure of `atp_block` is:

```
struct atp_block {
    struct sockaddr_at    *atp_saddr;       /* from/to address */
    union {
    struct sreq_st        sreqdata;         /* For send request. */
#define atp_sreqdata      atp_data.sreqdata.atpd_data
#define atp_sreqdlen      atp_data.sreqdata.atpd_dlen
#define atp_sreqtries     atp_data.sreqdata.atpd_tries
#define atp_sreqto        atp_data.sreqdata.atpd_to
     struct rres_st       rresdata;         /* for response. */
#define atp_rresiov   atp_data.rresdata.atpd_iov
#define atp_rresiovcnt atp_data.rresdata.atpd_iovcnt
     struct rreq_st       rreqdata;         /* for request data */
#define atp_rreqdata  atp_data.rreqdata.atpd_data
#define atp_rreqdlen  atp_data.rreqdata.atpd_dlen
     struct sres_st       sresdata;         /* for response */
#define atp_sresiov   atp_data.sresdata.atpd_iov
#define atp_sresiovcnt atp_data.sresdata.atpd_iovcnt
    } atp_data;
    u_char                atp_bitmap;       /* response buffer bitmap */
};
```

The following table describes the translator ATP Libatalk routines:

TABLE 1. Translator ATP Kernel Routines

Routine	Function	Internal Actions
`atp_open()`	Takes a socket (port number) used for listening for requests on that port, creates an `atp_handle` structure, and returns a pointer to it.	Opens a socket and performs a bind call to establish the socket.
`atp_rreq()`	Waits on an open `atp_handle` for a request from a specific address on a specific port. The data for the request arrive in the parameter block.	Calls `recv_atp` to receive request. `recv_atp` uses a `recvfrom` to wait for an input packet from the kernel.
`atp_rresp()`	Waits for a response on an open `atp_handle`. The data for the request is passed back in the parameter block.	
`atp_sreq()`	Sends a request on an open `atp_handle`. The parameters are received in a parameter block.	Builds an ATP request packet, and calls `sendto` to send it to the kernel.
`atp_sresp()`	Sends response packets on an open `atp_handle`. The parameters are received in a parameter block.	

4.2 ASP

Libatalk implements the following ASP routines: `asp_init`, `asp_getsession`, `asp_close`, `asp_attention`, `asp_getrequest`, `asp_cmdreply`, `asp_wrtcontinue`, `asp_wrtreply`, and `asp_kill`. The ASP structure (`include/atalk/asp.h`) contains the information about an ASP session. The structure of `ASP` is:

```
typedef struct ASP {
    ATP                  asp_atp;         /* The ATP information for this session. */
    struct sockaddr_at   asp_sat;         /* The source AppleTalk address for this
                                          session */
    u_char               asp_wss;
    union {
     struct {
     char                *as_status;
     int                 as_slen;
            }            asu_status;
    u_short              asu_seq;
          }              asp_u;
#define asp_status   asp_u.asu_status.as_status
#define asp_slen    asp_u.asu_status.as_slen
#define asp_seq    asp_u.asu_seq
    int                  asp_flags;
    u_char               asp_sid;         /* ASP session ID. */
} *ASP;
```

The following table describes the translator ASP Libatalk routines:

TABLE 2. Translator ASP Kernel Routines

Routine	Function
`asp_init()`	Allocates room for an ASP structure and adds the ATP structure passed to it as a parameter to the ASP structure it allocates.
`asp_getsession()`	Waits for and acts upon ASP control requests. If an `OpenSess` is sent by a Macintosh client, a child process is forked that handles that session. `asp_getsession` also manages sending status and replying to tickle packets.
`asp_close()`	Closes an open session.
`asp_attention()`	Sends an Attention message to the client, to inform it that the server needs attention.
`asp_getrequest()`	Receives a request from a client and allocates enough buffer space for the receipt of requests from the client.
`asp_cmdreply()`	Used to reply to commands received through `asp_getrequest`.
`asp_wrtcontinue()`	Transfers data from the server to the client as part of a write call sequence. The timeouts on the packets for `WrtContinue` are determined adaptively from the ATP layer Transaction Release delay times.

TABLE 2. Translator ASP Kernel Routines

Routine	Function
`asp_wrtreply()`	Terminates the write call chain.
`asp_kill()`	Sends a signal to all of the current sessions.

5. Information Table Maintenance

The AFP / AFS protocol translator maintains table information with the daemon `atalkd`. This daemon adds, deletes, and expires RTMP, NBP, and ZIP table entries. It also replies to incoming packets for AppleTalk Echo Protocol (AEP). Figure 5 shows the interaction between the components of `atalkd`.

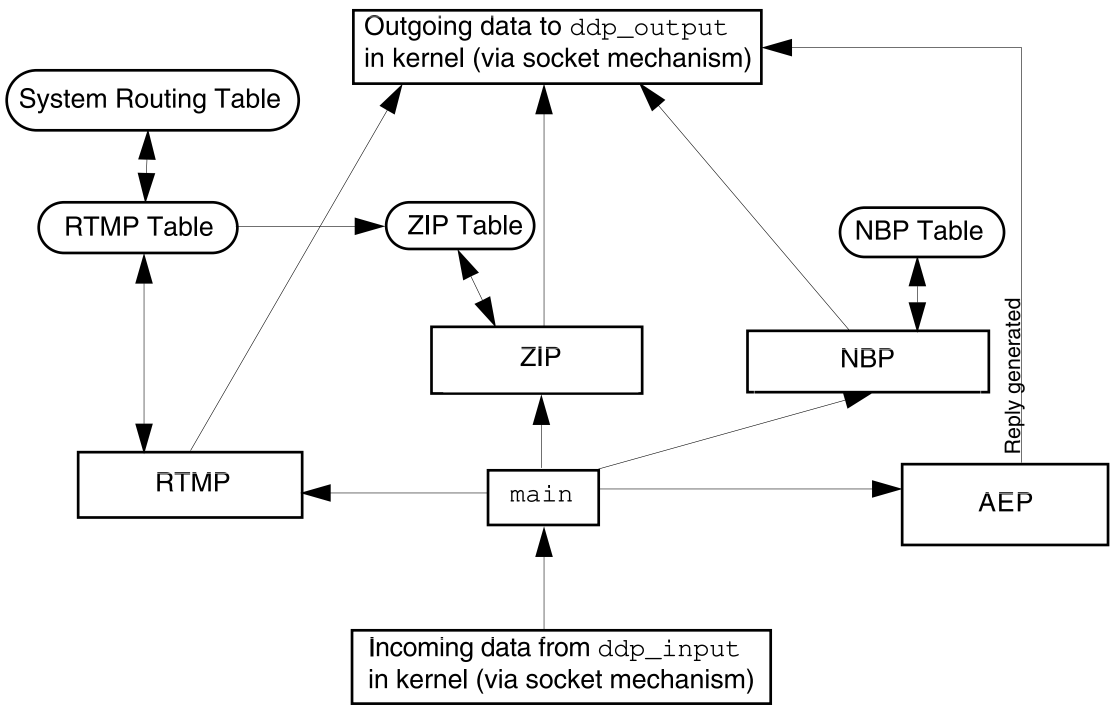


Figure 5. Information Table Maintenance (`atalkd`)

After parsing the command line arguments to `atalkd` and setting up the appropriate interfaces, `main` proceeds to open sockets for the DDP services `RTMP`, `NBP`, `AECHO`, and `ZIP`. The `main` routine calls the initialization routines in the supporting routines for each protocol, and then enters an infinite for loop that sleeps on a file descriptor, waiting to be woken by the socket layer, for the sockets it has open. When a packet is received for one of the protocols `main` handles, an appropriate routine is called and passed the socket from which to read.

6. AppleTalk Filing Protocol (AFP)

The AFP/AFS protocol translator provides a platform for AFP on UNIX. A daemon (`afpd`) runs, waiting for Macintosh users to connect to a file server advertised by a particular invocation of a daemon (i.e. one daemon per server advertised). When a user connects to a server and logs in, `afpd` forks a process to serve the user's requests. The requests come in as an index into a switch table that contains the pointer to an appropriate routine. When a volume is dismounted, the process created to serve the user terminates.

6.1 What is "AppleDouble"?

The Apple "AppleDouble" file format [9] addresses the issues that arise from the Macintosh using a typed file system whereas UNIX uses an untyped file system. Every Macintosh file has a set of attributes and resources associated with it that is separate from the data (hence the data and resource forks). UNIX has no mechanism for distinguishing between those attributes and the data in a file. Thus, the attributes and resources associated with a UNIX file must be stored in a separate file. The resource fork contains data used by an application, such as program code, icons, dialog boxes, and menus. The data fork of an application contains the data specific to that application.

For example, to store the application Microsoft Word on an AFP volume hosted on a UNIX system in your home directory, the data fork would be stored in your home directory under the name "Microsoft Word", and the resource fork would be stored (in the AFP implementation of the translator) in a subdirectory named ".AppleDouble" in your home directory. The name of the resource fork file would also be "Microsoft Word". Contained in every AFS directory that has Macintosh files with resource and data forks is an ".AppleDouble" directory that contains the resource forks for the files in that directory.

This naming convention creates a problem when you use the UNIX copy or move command to move a Macintosh file with both resource and data forks. If you simply copy an application from your home directory to another UNIX location, the resource fork is left behind, and attempts to launch the application will fail. Make sure that if you move the data fork, you also move the corresponding resource fork into the .AppleDouble directory of the target directory.

The resource fork file is formatted as follows:

Magic Number	4	For AppleDouble is 0x00051607
Version Number	4	0x00010000
Filler	16	
Number of Entries	2	Specifies the number of TYPES of entries
Entry descriptor for each entry		
Entry ID	4	Entry type identifier
Offset	4	Offset from the beginning of the file to the start of the date for the entry.
Length	4	Length of the entry data in bytes.

The possible entry types are:

Type Number	Type Name	Type Description
1	Data Fork	Data Fork
2	Resource Fork	Resource Fork
3	Real Name	The file's Macintosh name
4	Comment	Macintosh comment string
5	Icon, B&W	Black & White Icon
6	Icon, Color	Color Icon
8	File Dates Information	File date attributes
9	Finder Info	Macintosh Finder information
10	Macintosh File Information	Macintosh file information, attributes, etc.
13	Short Name	AFP short name
14	AFP File Information	AFP file information
15	Directory ID	AFP Directory ID

6.2 The Desktop Database

Another important component of the translator's AFP implementation is the Desktop database. The Desktop database is used by the Macintosh Finder to associate applications and documents with icons, to store the icon itself, to locate an appropriate application when a user opens a document, and to store comments for files and directories. The `afpd` daemon stores the Desktop database in a directory under the AFP volume mount point in a directory named ".AppleDesktop". The Desktop database entries are stored in subdirectories, named with the first letter of the CREATOR of the entry. Each CREATOR consists of two files: `CREATOR.appl` and `CREATOR.icon`.

The `.appl` file contains the path name of the application with which the CREATOR is associated, and also may hold a user-defined tag and a comment string. The `.icon` file contains the icon bitmaps (tagged by icon type) that are associated with the CREATOR. The Finder, indirectly, makes calls to the Desktop database to retrieve the information used to find the appropriate application for a document being opened, to get the bitmaps for icons, and to get the comment strings for files. The `afpd` daemon has a data structure to represent the currently open desktop entry that is used for representing the currently open `.icon` and `.appl` desktop files. Beyond that, there is no additional caching of this information.

For icons, the following data structure represents the current open `.icon` file (for a given CREATOR). For application mappings, this data structure represents the current open `.appl` file.

```
struct savedt {
    u_char      sdt_creator[ 4 ];     /* The CREATOR of the file. */
    int         sdt_fd;               /* The file descriptor to the open file. */
    int         sdt_index;            /* The index (1, 2, 3,...) icon or appl
                                         mapping in the file. */
    short       sdt_vid;              /* The volume identifier of the volume this
                                         goes with. */
};
```

6.3 Authentication

The Macintosh supports a mechanism called Alternative UAM (User Authentication Mechanism) that is used to do Kerberos authentication with the AFP server running on UNIX. The server returns an encrypted session key in response to the `LoginCont` call from the Macintosh. The Kerberos ticket is stored on the Macintosh in the `skey` resource in AFS Kerberos. Also, a utility named "AFS Log" can "refresh" an expired ticket, much as `klog` does on UNIX.

6.4 The AFP `main` Routine

The translator's AFP implementation has two large switch arrays that are indexed by the AFP function number. One array is preauthentication commands, the other is postauthentication commands. The entries in the arrays are function pointers to routines that read the data in the packet, perform the specified action, formulate a response, and return the response.

The `main` routine is the primary module that handles the dispatching of the functions. Upon entry (when `afpd` is started), it opens an ATP and ASP session, and registers the AFP server in the local NBP database. An infinite loop is then entered, waiting for an ASP command. If the ASP command is `ASP_CLOSE`, the session is closed. If the ASP function is another command, `main` reads the AFP command byte from the packet, and calls the appropriate function, with space allocated for the reply. The ASP reply is sent via `asp_cmdreply`. If the ASP command is `WRITE`, `main` reads the command byte, and calls the appropriate function. The response is sent via `asp_wrtreply`. Other ASP commands cause an error ("asp_getrequest: bad return") to be logged via `syslog`.

The function calls in `afpd` include functions grouped in the following categories: server, volume, directory, file, combined file and directory, fork, desktop database, and additional AFP calls. The `main` routine dispatches these calls via switch arrays in `switch.c`, where the arrays are indexed by the AFP call number. The functions `afp_login` and `afp_logincont` (located in the preauthentication switch array) may be called before authentication.

7. Printer Access Protocol (PAP)

A daemon (`papd`) runs, waiting for connections from a Macintosh. The daemon spools the data to a printer using the traditional UNIX print mechanisms, and sends print jobs to UNIX-connected PostScript [10] printers (figure 6).

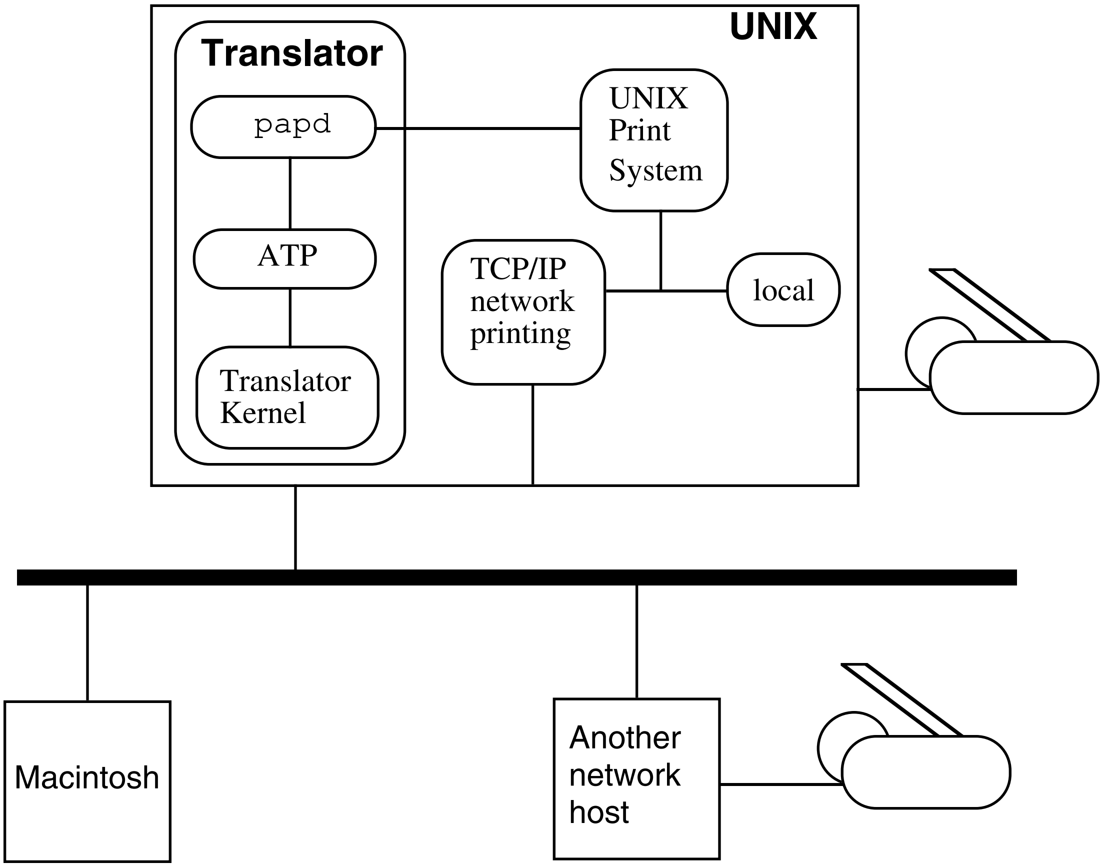


Figure 6. PAP Netatalk Implementation

7.1 The `main.c` Routine

When `papd` starts, it reads configuration information from the `papd.conf` file. It then opens an ATP listening socket and remains in an infinite processing loop where it first waits for a job (via `pap_getnextjob`), and then forks off a child process to process the job (via `process_job`). When the job is processed, that ATP socket is closed and the loop returns to `pap_getnextjob` where it waits for another connection (figure 7).

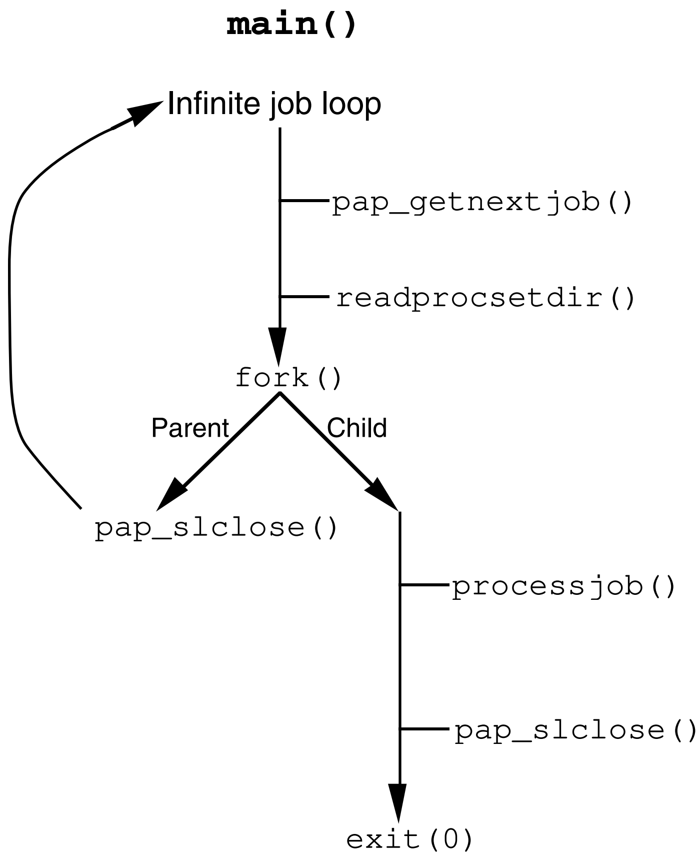


Figure 7. The `main.c` routine

8. Future Work

The IFS includes working protocol translators for the Macintosh and Sun's Network File System (NFS). The AFP/AFS protocol translator enables the largest segment of the campus computing community, Macintosh users, to use the IFS to easily share data and files with the rest of the UM campus. A new release of the AFP/AFS protocol translator is currently in beta test on the UM campus.

9. Acknowledgments

Wes Craig and Mark Smith wrote the original (phase I) version of Netatalk. Mike Stolarchuk ported the phase I code to IBM RS/6000 AIX. Marcus Watts provided helpful comments and improvements to the phase II work described here.

This work was supported by IBM.

10. References

1. Ted Hanss, "University of Michigan Institutional File System," */AIXTRA: The AIX Technical Review*, pp. 25–32 (January 1992).
2. Gursharan S. Sidhu, Richard Andrews, and Alan B. Oppenheimer, *Inside AppleTalk,* Second Edition, Addison-Wesley, Reading (1990).
3. Samuel J. Lefler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Addison-Wesley, Reading (1989).
4. J. H. Howard, "An Overview of the Andrew File System," pp. 23–26 in *Winter 1988 USENIX Conference Proceedings*, Dallas (February 1988).
5. Thomas Hacker, "Netatalk Architecture," Internal CITI document, The University of Michigan, August, 1992.
6. James Howe, "Intermediate File Servers in a Distributed File System Environment," CITI Technical Report 92–4, University of Michigan, (June 1992).
7. William Doster, "Kerberos User Authentication Method (UAM) for Use by Macintosh Users of Apple-Unix File Server (AUFS)," unpublished CITI internal document, February 1990.
8. William Doster, Jim Rees, "Third-Party Authentication in the Institutional File System," CITI Technical Report 92-1, University of Michigan, (February 1992).
9. "AppleSingle/AppleDouble Formats for Foreign Files Developer's Note," Apple Computer, Inc. (1990). Available from the Apple Product Developer's Association (APDA).
10. Ed Taft and Jeff Walden, *PostScript Language Reference Manual*, Second Edition, Addison-Wesley, Reading (1990), Appendix G, "Document Structuring Conventions—Version 3.0," pp. 611–708.

11. Availability

Copies of the translator may be requested from `info@citi.umich.edu`. The package will be distributed "as is," without CITI support.