New Technical Notes

Macintosh

Developer Support

Data Servers on AppleTalk

Networking

Revised by: March 1988
Written by: Bryan Stearns April 1985

Many applications could benefit from the ability to share common data between several Macintoshes, without requiring a file server. This technical note discusses one technique for managing this AppleTalk communication.

There are four main classes of network "server" devices:

Device Servers, such as the LaserWriter, allow several users to share a single hardware device; other examples of this (currently under development by third parties) are modem servers and serial servers (to take advantage of non-intelligent printers such as the ImageWriter).

File Servers, such as AppleShare, which support file access operations over the network. A user station sends high-level requests over the network (such as "Open this file," "Read 137 bytes starting at the current file position of this file," "Close this file," etc.).

Block Servers, which answer to block requests over the network. These requests impart no file system knowledge about the blocks being passed, i.e., the server doesn't know which files are open by which users, and therefore cannot protect one user's open file from other users. Examples of this type of server are available from third-party developers.

Data Servers, which answer to requests at a higher level than file servers, such as "Give me the first four records from the database which match the following search specification." This note directs its attention at this type of server.

A data server is like a file server in that it responds to intelligent requests, but the requests that it responds to can be more specialized, because the code in the server was written to handle that specific type of request. This has several added benefits: user station processing can be reduced, if the data server is used for sorting or searching operations; and network traffic is reduced, because of the specificity of the requests passed over the network. The data server can even be designed to do printing (over the network to a LaserWriter, or on a local ImageWriter), given that it has the data and can be directed as to the format in which it should be printed.

ATP: The AppleTalk Transaction Protocol

ATP, the assured-delivery AppleTalk Transaction Protocol, can be used to support all types of server communications (the LaserWriter uses ATP for its communications!). Here is a possible scenario between two user stations (“Dave” and “Bill”) and a data server station (“OneServer”,

a server of type "MyServer"). We've found that the "conversational" analogy is helpful when planning AppleTalk communications; this example is therefore presented as a conversation, along with appropriate AppleTalk Manager calls (Note that no error handling is presented, however; your application should contain code for handling errors, specifically the "half-open connection" problem described below).

Establishing the Connection

Each station uses `ATPLoad` to make sure that AppleTalk is loaded. The server station, since it wants to accept requests, opens a socket and registers its name using `NBPRegister`. The user stations use `NBPLookUp` to find out the server's network address. This looks like this, conversationally:

Server:	"I'm ready to accept requests!"	`ATPLoad`	Opens AppleTalk
		`OpenSocket`	Creates socket
		`NBPRegister`	Assigns name to socket
		`ATPGetRequest`	queue a few asynchronous
		`ATPGetRequest`	calls, to be able to handle several
		`ATPGetRequest`	users
Dave:	"Any 'MyServers' out there?"	`ATPLoad`	Opens AppleTalk
		`NBPLookup`	look for servers, finds OneServer
Dave:	"Hey, MyServer! What socket should I talk to you on?"	`ATPRequest`	Ask the server which socket to use for further communications
Bill:	"Any 'MyServers' out there?"	`ATPLoad`	Opens AppleTalk
		`NBPLookup`	look for servers, finds OneServer
Bill:	"Hey, MyServer! What socket should I talk to you on?"	`ATPRequest`	Ask the server which socket to use for further communications
Server:	"Hi, Dave! Use Socket N."	`ATPOpenSkt`	Get a new socket for talking to Dave
		`ATPResponse`	Send Dave the socket number
		`ATPGetRequest`	Replace the used GetRequest
Server:	"Hi, Bill! Use socket M."	`ATPOpenSkt`	Get a new socket for talking to Bill
		`ATPResponse`	Send Bill the socket number
		`ATPGetRequest`	Replace the used GetRequest

From this point on, the server knows that any requests received on socket N are from Dave, and those received on socket M are from Bill. The conversations continue, after a brief discussion of error handling.

Half-Open Connections

There is a possibility that one side of a connection could go down (be powered off, rebooted accidently, or simply crash) before the connection has been officially broken. If a user station goes down, the server must throw away any saved state information and close that user's open socket. This can be done by requiring that the user stations periodically "tickle" the server:

every 30 seconds (for example) the user station sends a dummy request to the server, which sends a dummy response. This lets each side of the connection know that the other side is still “alive.”

When the server detects that two intervals have gone by without a tickle request, it can assume that the user station has crashed, and close that user’s socket and throw away any accumulated state information.

The user station should use a vertical-blanking task to generate these tickle requests asyncronously, rather than generating them within the `GetNextEvent` loop; this avoids problems with long periods away from `GetNextEvent` (such as when a modal dialog box is running). This task can look at the time that the last request was made of the server, and if it’s approaching the interval time, queue an **asynchronous** request to tickle the server (it’s important that any AppleTalk calls made from interrupt or completion routines be asynchronous).

If a user station’s request (including a tickle request) goes unanswered, the user station should recover by looking for the server and reestablishing communications as shown above (beginning with the call to `NBPLookUp`).

More information about half-open connections can be found in the Printer Access Protocol chapter of *Inside LaserWriter,* available from APDA.

Using the Connection

The user stations Dave and Bill have established communications with the server, each on its own socket (note that the user stations have not had to open their own sockets, or register names of their own, to do this—the names we’re using are merely for explanational convenience). They are also automatically tickling the server as necessary.

Now the user stations make requests of the server as needed:

Bill:	“I’d like to use the sales figures for this year.”	`ATPRequest`	Bill opens a database.
Server:	“Ok, Bill.”	`ATPResponse`	The server checks to make sure that no one else is using that database.
Dave:	“Hey, Server - I’m still here!”	`ATPRequest`	Dave notices that the interval time is approaching, and makes a tickle request.
Server:	“Ok, Dave.”	`ATPResponse` from Dave”.	The server resets its “last time I heard
Bill:	“Please print the figures for January thru June.”	`ATPRequest`	Bill asks for specific data.
Server:	“Ok, Bill.”	`ATPResponse`	The server does a database search sorts the results, and prints them on a local Imagewriter.

Dave:	"I'd like to use the sales figures for this year."	`ATPRequest`	Dave opens a database.
Server:	"Sorry, Dave, I can't do that. Bill is using that database."	`ATPResponse`	The server finds that Bill is using that data.

Closing the Connection

The user stations continue making requests of the server, until each is finished. The type of work being done by the server determines how long the conversation will last: since the number of sockets openable by the server is limited, it may be desirable to structure the requests in such a way that the average conversation is very short. It may also be necessary to have a (NBP named) socket open on the user station, if the server needs to communicate with the user on other than a request-response basis. Here is how our example connections ended:

Dave:	"Thank you, server, I'm done now. You've been a big help."	`ATPRequest`	Dave tells the server he's finished.
Server:	"Ok, Dave. Bye now."	`ATPResponse` T	the server kisses Dave goodbye.
		`ATPCloseSkt`	After the Response operation completes, the server closes the socket Dave was using. It also
	`ATPCloseSkt`	notices that Bill hasn't sent a request in more than two intervals, and closes Bill's socket, too.	

The user station can forget about the socket it was using on the server; if it needs to talk with the server again, it starts at the `NBPLookUp` (just in case the server has moved, gone down and come up, etc.).

Further Reference:

- The AppleTalk Manager *Inside LaserWriter*