Technical Note NW560

Macintosh Protocol Package Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic - questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Oct 01 1990]

Retries and tickle mechanism for nonAppleShare connections

Date Written: 12/28/92

Last reviewed: 3/1/93

We're experiencing time-outs in the .MPP driver communications between our non-AppleShare server and applications that talk to it, when the connection is low-speed (for example, an AppleTalk Remote Access connection). We have no problems over local area nets or with AppleShare connected over our low-speed connection. What values should we use for the ATP timeOutVal and retryCount?

The reason you don't see problems with AppleShare connections is that AppleShare uses the AppleTalk Filing Protocol (AFP) and AFP is built on the AppleTalk Session Protocol (ASP). ASP uses an infinite retry count on its ATP transactions and a tickle mechanism to detect broken session connections. If ASP's tickle mechanism finds the session connection is dead, it kills all outstanding asynchronous ATP calls with PKillSendReq and PKillGetReq. ASP clients don't have to worry about time-out conditions except when the network connection fails.

The AppleShare workstation determines the timeOutVal to pass to AFP/ASP by using the Apple Echo Protocol. Here's how you could use the echo protocol to determine the timeOutVal for your program:

1. Send the other end of the new connection a full-size (586-byte) packet to the echo socket.
2. Wait for up to 10 seconds for the echo packet to come back. This should be enough time to send a full-size packet both ways at 1200 bps (slower than any connection you'd hope to encounter on an AppleTalk network).
3. If the echo packet isn't back in 10 seconds, assume the packet was lost either going to or coming from the other end and use a default retry time of 4 seconds (this is what AppleShare uses as a default when its echo packet is lost--it's generally a good value for LocalTalk).
4. If the echo packet is back within 10 seconds, multiply the turnaround time by 4.5 (enough time to transfer nine full-size packets).
5. Add the maximum amount of latency you expect the responding side of the connection to have between receiving a request and sending the response (you'll have to determine this value by running your code on a slow system).
6. If the calculated value ((echo time * 4.5) + latency) is greater than 2 seconds, use it. Otherwise use a minimum default retry time of 2 seconds. The reason for keeping the minimum above a certain level is to minimize network traffic.

We recommend building a session protocol something like ASP on top of ATP so that you can use infinite retries and let a tickle mechanism tear down a broken connection. A tickle mechanism is also easy to implement. For example, to implement a tickle mechanism like the one AppleTalk protocols use, just start a VBL task with a vblCount of 2 minutes (120 ticks) on both ends of the connection. Every time you receive a packet of any kind from the other side of the connection, reset the vblCount to 2 minutes again. If the VBL task ever gets called, the connection was broken. To send the tickle packets, just

call PSendRequest (or PNSendRequest) to send a small tickle request with an infinite retry count and a retry period of 30 seconds. Whenever you receive a tickle packet from the other side of a connection, simply ignore it (if you don't respond, ATP will keep retrying every 30 seconds, which is what you want).

If you don't want to use a tickle mechanism, you'll have to decide how many times you want ATP to retry. The value you use should balance how long you want a user to wait for something to happen and how reliable the user's network is at delivering packets -- something that isn't at all easy to determine.

For more information, see the Macintosh (Networking) Technical Note "AppleTalk Timers Explained."

Back to top

MPP stands for Macintosh Protocol Package handler

Date Written: 2/28/92

Last reviewed: 2/28/92

What does "MPP" stand for? We know it's a lower-level driver that contains code to implement ALAP (Link Access Protocol), DDP (Datagram Delivery Protocol), NBP (Name Binding Protocol), and RTMP (Routing Table Maintenance Protocol) stuff, but we're curious to the point of not being able to do any work :) Macintosh Packet Protocol? Multiple Problem Protocol?

MPP stands for Macintosh Protocol Package handler. XPP stands for eXtended Protocol Package handler; ATP stands for AppleTalk (originally AppleBus) Transaction Protocol handler; DSP is Data Stream Protocol handler; LTM is Link Tool Manager; and MNP is Microcom Networking Protocol, implemented by AppleTalk Remote Access across the remote link. Hope you can get some work done now ;)

Back to top

RAM-based AppleTalk driver fixes Mac Plus PNSendRequest error

Date Written: 1/1/91

Last reviewed: 1/1/91

I get an odd address error when using PNSendRequest on the Macintosh Plus. The Macintosh Plus has version 19 of the .MPP driver. Is there is an INIT or something that patches the older Macintosh models with the newer drivers?

Inside Macintosh suggests checking the .MPP driver version to see if the new driver is available (version >= 48).

You need the RAM-based AppleTalk drivers on the older Macintosh models to use the calls documented in *Inside Macintosh* Volume V. The RAM-based AppleTalk driver is available as an AppleTalk System file, and you need to place it in the System Folder of your Mac Plus.

X-Ref:

"The AppleTalk Manager," *Inside Macintosh* Volume V

Back to top

PKillNBP aKillQEl pointer

Date Written: 6/12/92

Last reviewed: 9/15/92

Where can I get the Q element pointer for a PLookupName call in order to call PKillNBP?

The pointer that you pass in the parameter block to PKillNBP, is the pointer to the parameter block of the outstanding lookup request. If several lookup requests have been made asynchronously, and are outstanding, then you would take the pointer to each parameter block and issue separate PKillNBP calls for each lookup to kill the request on.

Each asynchronous lookup request along with other asynchronous requests to the drivers get queued for processing. The KillNBP request takes the pointer to the parameter block of the outstanding request and searches through the driver queue, looking for a match. When a match is found, the request is dequeued and the driver notified of the action. If the request has already been dequeued--for example, if the original lookup request was completed while the PKillNBP was made--a cbNotFound error (error -1102) will be returned.

Back to top

Downloadables

Acrobat version of this Note (K) Download

Back to top
