New Technical Notes

Macintosh

Developer Support

Datagram Delivery Protocol Q&As

Networking

Revised by: Developer Support Center October 1992
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

Datagram Delivery Protocol (DDP) header checksum field
Date Written: 4/2/91
Last reviewed: 5/28/91

Does the Macintosh client pay attention to and validate Datagram Delivery Protocol (DDP) checksums within long DDP packets received?

—

DDP is a client of the AppleTalk Transaction Protocol (ATP), which is a client of the AppleTalk Session Protocol (ASP) and AppleTalk Filing Protocol (AFP). DDP is told by the entity for whom it is a client whether it should use checksumming. For performance reasons, our version of ATP does not specify that DDP use the checksum. Both AppleShare and the AppleShare client are out of the loop; neither can specify whether or not to use the DDP checksum.

The LAP layer performs checksumming, ensuring no on-the-wire corruption. The only benefit of DDP checksumming is end-to-end corruption detection when the packet has travelled over a malicious router that has validly received a DDP packet on one net, changed the packet, re-computed the LAP checksum, and sent it out on another cable. Because of the performance hit of DDP checksumming and the rarity of such a malicious router, it was

decided not to request DDP checksumming.

ZIP Query packet zero checksum value
Date Written: 3/3/92
Last reviewed: 4/22/92

Is a zero checksum value indicative of no checksum provided with the Zone Information Protocol (ZIP) packet?

—

True, a zero checksum in any DDP packet with an extended header (including ZIP packets) indicates that a checksum was not performed. See *Inside AppleTalk,* 2nd Ed. pages 4-17.

How to write a DDP read with completion routine
Date Written: 2/28/92
Last reviewed: 2/28/92

The PWriteDDP call is nothing like a PReadDDP call. We want to do an AppleTalk Datagram Delivery Protocol (DDP) read with a completion routine (to wait for an echo reply), but with only the old style call, this is impossible. Is a completion routine for DDPRead possible?

—

Because of the ABRecord structure handling implemented in the DDPRead call, an asynchronous version of this call was not possible. DDPRead implements Memory Manager calls. As a result, no preferred style of call was implemented for DDPRead. The alternative is to alter your process to use the POpenSkt call instead. As part of the POpenSkt function, you'll need to define a socket listener. A sample socket listener can be found in the Tech Note "AppleTalk Timers Explained" (it’s written in MPW assembly code). In addition, a sample program called DMZ, on the System 7 Golden Master CD, uses that echo socket listener code fragment.

Communication between players in a game on an AT network
Date Written: 12/18/90
Last reviewed: 2/1/91

How can I use the broadcast mode in an AppleTalk DataGram Delivery Protocol (DDP) layer to communicate with the other players of a game that is on a network and not interfere with other network users?

—

The broadcast mode may seem like a solution, but Apple doesn’t recommend using broadcast mode. Applications like games generally end up sending LOTS of update packets to other players of the games. If your program broadcasts those packets, then all nodes on your network will be interrupted for every packet sent and that can cause performance

problems on every machine on a network. In Apple supplied software, broadcast mode is only used when packets need to be sent to every node on a network. For example, Name Binding Protocol (NBP) Lookup packets are broadcast because the network address is unknown.

The following suggestions should help implement communications between players.

The DDP protocol layer does not guarantee delivery of packets (the disadvantage), but has very little overhead (the advantage). If you require delivery of all packets and can take a small additional performance hit, you should use the AppleTalk Transaction Protocol (ATP) layer.

The DDP type you use must be in the range of $10-$FF. The DDP type you choose to use (in that range) probably won't make a lot of difference unless you define multiple clients to DDP within your socket listener. Packets addressed to your socket will get sent to your socket listener. How you use the type is up to you (as long as you stay out of the reserved range).

You should open a dynamic socket (i.e., use 0 for the socket number and use the socket number in the range $80-$FE that is returned) with the OpenSocket call and then register a name on the network for that socket with the RegisterName call. If all copies of the game register with the same NBP type, then you can use the LookupName call to find other players on the network. Once you've found other players and decided who you're going to play against, you just address each DDP packet to the internet address(es) of the other player(s) socket(s). Several multi-player network games I've seen on the Macintosh do just that. (Hint: If you limit games to players on a single network, DDP will use short header packets which are slightly faster to send (8 bytes less per packet).)

If you build a table (array) of network addresses from those playing a game, you can use this algorithm to send a packet to each player.

```
BEGIN
 { put data into buffer for DDP and set the common parameter block fields: }
 { Async Flag, Command, Checksum Flag, Source Socket, DDP Type, BDS Pointer }
 { and Destination Network (if limiting play to a single network). }
 buildPacket;

 FOR Player := 1 to NumberOfPlayers DO
   BEGIN
     { Set the Destination Node and Destination Socket fields of the }
     { parameter block to the network address of a player.  That address }
     { would be in an array of network addresses (one for each player). }
     setAddress(Player);

     { Send the datagram to a player using SendDatagram call. }
     doSendDatagram;
   END;
END;
```