New Technical Notes
Macintosh

Developer Support

AppleTalk Timers Explained

Networking

Written by: Sriram Subramanian & Pete Helme April 1990

This Technical Note explains how to effectively use timers and retry mechanisms of the various AppleTalk protocols to achieve maximum performance on an internet.

The most fundamental service in an AppleTalk internet is the Datagram Delivery Protocol (DDP), which provides a best-effort, connectionless, packet delivery system. A sequence of packets sent using DDP on an AppleTalk internet between a pair of machines may traverse a single high-speed Ethernet network or it may wind across multiple intermediate data links such as LocalTalk, TokenRing, etc., which are connected by routers. Some packet loss is always inevitable because of the loosely coupled nature of the underlying networks. Even on a single high-speed Ethernet network, packets can be lost due to collisions or a busy destination node. The AppleTalk Transaction Protocol (ATP), the AppleTalk Data Stream Protocol (ADSP), and other high-level protocols protect against packet loss and ensure reliability by using positive acknowledgement with packet retransmission mechanism.

The basic transaction process in ATP consists of a client in a requesting node sending a Transaction Request (TReq) packet to a client in a responding node. The client in the responding node is expected to service the request and generate a series of Transaction Response (TResp) packets, which also serves as an acknowledgement. The ATP process in the requesting node also starts a timer when it sends a packet and retransmits a packet if the timer expires before a complete response arrives. In a large internet with multiple gateways, it is impossible to know how quickly acknowledgements may return to the requestor. If you set the retry time to be too small, you may be retransmitting a request while a delayed response is *en route*, but if you wait too long to retransmit a request, application performance may suffer. More importantly, the delay at each gateway depends upon the traffic, so the time required to transmit a packet and receive an acknowledgement varies from one instant to another. To further complicate matters, two packets sent back to back could take completely different routes to the destination.

Selecting ATP Retry Time And Retry Count

You can use the round trip time for a transaction as a heuristic for setting the retry time and retry count. The round trip time between two nodes in a particular internet at a particular time is usually deterministic.

The easiest way to set the retry time is to assign a static value based on the round trip time for a transaction. The AppleTalk Echo Protocol (AEP) can be used to obtain the round trip time in a given moment between two nodes. AEP is implemented in each node as a DDP client residing on statically-assigned socket number four. You should use DDP to send AEP requests through any socket that is available, and you should use the maximum packet size that you plan on using in your application. You can listen for AEP responses by implementing a socket listener. The following code is an example AEP socket listener.

```
;__________________________________________________________________________
;__________________________________________________________________________
;
; EchoDude
;
; 3/90 pvh - MacDTS
;
; ©1990 Apple Computer, Inc.
;
; The following MPW Asm code is a socket listener for reading in returned Echo
; (DDP type 4) packets.
;
; The target device was shipped a packet with a '1' in the first byte of the data
; area by way of a DDPWrite.  It was sent to socket 4, the Echoer socket.  If the
; target device has an Echoer, it will send a return packet to us of equal size
; except it will have replaced the '1' in the first byte with the value '2'.  This
; indicates an EchoReply packet.
;
; The listener itself (RcvEcho) is added with a POpenSkt (Inside Mac V-513) call by
; passing the address of the listener in the listener field of the parameter block.
;
; All we really are trying to accomplish here is to set up a notification for
; returned packets from the target Echoer.  A time (Ticks) is stuffed into a
; location our app can find (actually back into the packet buffer) and will be used
; to calculate round trips times.  We'll also save off the hop count from the packet
; header for fun too since I have nothing better to do with my time on weekends.
;
; More could be done with this listener as far as making sure that we are only
; receiving back a packet from the node we sent it to etc.... but we can't
; encompass everything in a sample.  Okay, well we could… but we have to leave
; something for you guys to do.
;
; It should be noted that careful preservation of register A5 is necessary.
; LAP requires that A5 be preserved AFTER the call to ReadRest. i.e. you
; cannot save A5 onto the stack when your socket listener is entered, call ReadRest
; and then restore A5 from the stack and exit.  Wah.  LAP requires that the address
; placed in A5 during ReadRest be there when your socket listener is exited.
; So… if you need a different A5 after the call to ReadRest make sure you restore
; it before RTS-ing back the caller.
;
;
;    Called:
;         A0,A1,D1 : Preserve until after ReadRest
;         A2 -> MPP local variables
;         A3 -> RHA after DDP header
;         A4 -> ReadPacket, 2(A4) -> ReadRest
;         A5 Useable until ReadRest
;         A6,D4-D7 : Preserve across call
;
;___________________________________________________________________________

EchoSkt    EQU    4        ; Echo socket number
EP         EQU    4        ; EP DDP protocol type
EPReq      EQU    1        ; Code for echo request
EPReply    EQU    2        ; Code for echo reply

;
; Read the packet into the echo buffer
```

```
;

RcvEcho    PROC    EXPORT
                  EXPORT    our_A5 : CODE
                  EXPORT    our_Buff : CODE
                  IMPORT    GBOB:DATA

       BRA.S     checkEcho
our_A5
       DC.L      0
our_Buff
       DC.L      0
our_Hops
       DC.W      0
our_Ticks
       DC.L      0
checkEcho
       CMP.B     #EP,-(A3)            ; Make sure it's an echo packet
       BNE.S     RcvEIgnore           ; Ignore it if not
       LEA       toRHA(A2), A3        ; top of RHA
       CLR.L     D2                   ; clean up D2
       MOVE.B    lapType(A3), D2      ; lap type
       CMP.B     #longDDP, D2         ; check for long header (Type #2 packet)
       BNE.S     noHops               ; wah... no hops if short packet
       MOVE.B    lapType+1(A3), D2    ; this is the hop count byte, 1st byte in DDP
                                     ; header
       AND.B     #$3C, D2             ; mask to middle 4 bits of byte for hop count
                                      ;     | x | x | H | O | P | S | x | x |
       ASR.B     #2, D2               ; shift 2 bits to right
       LEA       our_Hops, A3         ; address of our storage
       MOVE.B    D2, (A3)             ; move # of hops into our storage
noHops
       MOVE.W    #DDPMaxData, D3      ; our buffer is #DDPMaxData in size
       LEA       our_Buff, A3         ; address of buffer to read packet into
       MOVE.L    (A3), A3             ; set buffer
       JSR       2(A4)                ; ReadRest of packet into buffer
       BEQ.S     RcvEchoReply         ; If no error, continue
       BRA.S     RcvEchoFail          ; dang…
RcvEIgnore
       CLR       D3                   ; Set to ignore packet
       JMP       2(A4)                ; Ignore it, ReadRest and return
       BRA.S     RcvEchoFail
RcvEchoReply
       CMP.B     #EPReply, -DDPMaxData(A3)    ; make sure it's our reply packet
                                      ; it shouldn't be anything else, but check
                                      ; just in case
       BNE.S     RcvEchoFail          ; if not our reply then blow
       MOVE.L    A5, D2               ; save dude in D2
       LEA       our_A5, A5           ; address of our A5 local storage
       MOVE.L    (A5), A5             ; make A5 our A5 for application global use
       MOVE.B    #1, GBOB(A5)         ; set flag confirming reception of
                                      ; echo reply packet
       LEA       our_Buff, A3         ; address of our local buffer storage into A3
       MOVE.L    (A3), A3             ; get saved pointer and set buffer.
       LEA       our_Hops, A5         ; address of hops local storage… notice we
                                      ; are TRASHING A5 with this!!!!!
       MOVE.W    (A5), (A3)+          ; copy in hop count to buffer
       MOVE.L    Ticks, (A3)          ; next copy in Ticks

       MOVE.L    D2, A5               ; restore dude
       RTS                            ; return to caller
RcvEchoFail
       RTS                            ; return to caller
       ENDP

setUpSktListener    PROC    EXPORT
                           IMPORT    our_A5 : CODE
                           IMPORT    our_Buff : CODE

       LEA       our_A5, A0           ; this copies
       MOVE.L    CurrentA5, (A0)      ; this copies CurrentA5 into our local
                                      ; storage for global use in the listener
       MOVE.W    #DDPMaxData, D0      ; max size of data in a packet
       _NewPtr   CLEAR
       BNE.S     setUpFailed          ; if NIL then forget it

       LEA       our_Buff, A1         ; we need to save the pointer reference
       MOVE.L    A0, (A1)             ;  in a place the listener can find it
       MOVE.L    A0, D0               ; return value to caller
       RTS
setUpFailed
       CLR.L     D0                   ; tell caller we failed by returning nil
                                      ; (caller expecting valid ptr returned)
       RTS

       ENDP

       END
```

We now resume our regular programming…

You should typically get an AEP response packet within a few milliseconds. If there is no response for a period of time, typically about 10 seconds, you should resend your AEP request to account for a lost request or lost packets. To be really safe, you should resend your AEP request with different data to take into account the response to the first packet coming back later. The retry time could then be simply set to `k*Round_Trip_Time`, where the value of `k` depends upon the request semantics, like total data size.

This technique of statically setting the retry time is not always adequate to accommodate the varying delays encountered in a internet environment at different times. You could dynamically adjust the retry time based on an adaptive retransmission algorithm that continuously monitors round trip times and adjusts its timeout parameter accordingly. To implement an adaptive algorithm, you can record the round trip time for each transaction. One common technique is to keep the average round trip time as a weighted average and use new round trip times from transactions to change the average slowly. For example, one averaging technique* uses a constant weighing factor, `q`, where $0 \leq q < 1$, to weigh the oldest average against the latest round trip time:

```
   W_aver = (q * W_aver ) + (( 1 - q) * New_Round_Trip_Time)
```

Choosing a value for `q` close to 1 makes the weighted average immune to changes that last a short time. Choosing a value for `q` close to 0 makes the weighted average respond to changes in the delay very quickly.

The total time (i.e., `retry time * retry count)` before a request is concluded as failed could be anywhere from 10 seconds to a couple of minutes, depending on the type of the client application and the relative distance between the source and the destination.

*Douglass Corner, InterNetworking with TCP/IP. KARN, P. and C. PARTRIDGE [August 1987], “Improving Round-Trip Time Estimates in Reliable Transport Protocols”, *Proceedings of ACM SIGCOMM 1987*.

NBP Retry Counts

You cannot really use the AEP to estimate round trip times for NBP packets because you need to use NBP to determine the internet address of the node from which an echo is being sought. In this case, you have to use the type of device that you are looking for as a heuristic for setting the retry count. The LaserWriter, for example, may be busy and not respond to a LkUp packet. In such a case, you might want to do a quick lookup to return a partial list to the user like the Chooser. You could then do a longer lookup to get a more complete list of mappings. You should use a “back off” algorithm to make the subsequent lookups further apart to generate progressively less traffic. Name lookups are expensive and produce a lot of network traffic, and name confirmation is the recommended call to use when confirming mappings obtained through early bindings. Because Name lookups are expensive, you should avoid searching all the zones in the internet.

Setting TRel Timer in SendRequest

AppleTalk Phase 2 drivers allow you to set the TRel timer in `SendRequest` or `NSendRequest` calls with ATP XO (exactly once) service so as not to be locked into the pre-AppleTalk Phase 2 time of 30 seconds. You should set this timer based on the round trip time. Generally, if the round trip time is less than one second, the default TRel time setting of 30 seconds is adequate. If the round trip time is more, you can increase the TRel time proportionately.

xppTimeout and xppRetry

The two ZIP calls, `GetZoneList` and `GetLocalZones`, made on the .XPP driver contain the ATP retry interval (in seconds) and count, in the `xppTimeout` and `xppRetry` parameters. Both these functions are ATP request-response transactions between a node and a router on the network to which the requesting node is attached. The round trip is relatively short for this transaction, and you should have very small values of `xppTimeout` and `xppRetry,` typically two and three, respectively.

Further Reference:

- Inside AppleTalk
- Inside Macintosh, Volumes II & V, The AppleTalk Manager
- M.NW.Internets
- M.NW.AppleTalk2Mac