AppleTalk Transaction Protocol Basic interactions

This section covers ATP in greater depth, providing more detail about three of its fundamental concepts: transactions, buffer allocation, and recovery of lost datagrams.

Transactions

A transaction is an interaction between two ATP clients, known as the requester and the responder. The requester calls the .ATP driver in its node to send a transaction request (TReq) to the responder, and then awaits a response. The TReq is received by the .ATP driver in the responder's node and is delivered to the responder. The responder then calls its. ATP driver to send back a transaction response (TResp), which is received by the requester's .ATP driver and delivered to the requester.

Simple examples of transactions are:

- read a counter, reset it and send back the value read
- read six sectors of a disk and send back the data read
- write the data sent in the TReq to a printer

A basic assumption of the transaction model is that the amount of ATP data sent in the TReq specifying the operation to be performed is small enough to fit in a single datagram. A TResp, on the other hand, may span several datagrams, as in the second example. Thus, a TReq is a single datagram, while a TResp consists of up to eight datagrams, each of which is assigned a sequence number from 0 to 7 to indicate its position in the response.

The requester must, before calling for a TReq to be sent, set aside enough buffer space to receive the datagrams(s) of the TResp. The number of buffers allocated (in other words, the maximum number of datagrams that the responder can send) is indicated in the TReq by an eight-bit bit map. The bits of this bit map are numbered 0 to 7 (the least significant bit being number 0); each bit corresponds to the response datagram with the respective sequence number.

Datagram Loss Recovery

The way that ATP recovers from loss situations is best explained by an example. Assume that the requester wants to read six sectors of 512 bytes each from the responder's disk. The requester puts aside six 512-byte buffers (which may or may not be contiguous) for the response datagrams, and calls ATP to send a TReq. In this TReq the bit map is set to binary 00111111 or decimal 63. The TReq carries a 16-bit transaction ID, generated by the requester's .ATP driver before sending it. (This example assumes that the requester and responder have already agreed that each buffer can hold 512 bytes.) The TReq is delivered to the responder, which reads the six disk sectors and sends them back, through ATP, in TResp datagrams bearing sequence numbers 0 through 5. Each TResp datagram also carries exactly the same transaction ID as the TReq to which they're responding.

There are several ways that datagrams may be lost in this case. The original TReq could be lost for one of many reasons. The responding node might be too busy to receive the TReq or might be out of buffers for receiving it, there could

be an undetected collision on the network, a bit error in the transmission line, and so on. To recover from such errors, the requester's .ATP driver maintains an ATP retry timer for each transaction sent. If this timer expires and the complete TResp has not been received, the TReq is retransmitted and the retry timer is restarted.

A second error situation occurs when one or more of the TResp datagrams is not received correctly by the requester's .ATP driver. Again, the retry timer will expire and the complete TResp will not have been received; this will result in a retransmission of the TReq. However, to avoid unnecessary retransmission of the TResp datagrams already properly received, the bit map of this retransmitted TReq is modified to reflect only those datagrams not yet received. Upon receiving this TReq, the responder retransmits only the missing response datagrams.

Another possible failure is that the responder's .ATP driver goes down or the responder becomes unreachable through the underlying network system. In this case, retransmission of the TReq could continue indefinitely. To avoid this situation, the requester provides provides a maximum retry count; if this count is exceeded, the requester's .ATP driver returns an appropriate error message to the requester.

> **Note:** There may be situations where, due to an anticipated delay, you'll want a request to be retransmitted more than 255 times; specifying a retry count of 255 indicates "infinite retries" to ATP and will cause a message to be retransmitted until the request has either been serviced, or been cancelled through a specific call.

Finally, in our example, what if the responder is able to provide only four disk sectors (having reached the end of the disk) instead of the six requested? To handle this situation, there's an end-of-message (EOM) flag in each TResp. In this case, the TResp datagram numbered 3 would come with this flag set. The reception of this datagram informs the requester's .ATP driver that TResps numbered 4 and 5 will not be sent and should not be expected.

When the transaction completes successfully (all expected TResp datagrams are received or TResp datagrams numbered 0 to *n* are received with datagram *n*'s EOM flag set), the requester is informed and can then use the data received in the TResp.

ATP provides two classes of service: at-least-once (ALO) and exactly-once (XO). The TReq datagram contains an XO flag that's set if XO service is required and cleared if ALO service is adequate. The main difference between the two is in the sequence of events that occurs when the TReq is received by the responder's .ATP driver.

In the case of ALO service, each time a TReq is received (with the XO flag cleared), it's delivered to the responder by its . ATP driver; this is true even for retransmitted TReqs of the same transaction. Each time the TReq is delivered, the responder performs the requested operation and sends the necessary TResp datagrams. Thus, the requested operation is performed at least once, and perhaps several times, until the transaction is completed at the requester's end.

The at-least-once service is satisfactory in a variety of situations--for instance, if the requester wishes to read a clock or a counter being maintained

at the responder's end. However, in other circumstances, repeated execution of the requested operation is unacceptable. This is the case, for instance, if the requester is sending data to be printed at the responding end; exactly-once service is designed for such situations.

The responder's .ATP driver maintains a transactions list of recently received XO TReqs. Whenever a TReq is received with its XO flag set, the driver goes through this list to see if this is a retransmitted TReq. If it's the first TReq of a transaction, it's entered into the list and delivered to the responder. The responder executes the requested operation and calls its driver to send a TResp. Before sending it out, the .ATP driver *saves* the TResp in the list.

When a retransmitted TReq for the same XO transaction is received, the responder's .ATP driver will find a corresponding entry in the list. The retransmitted TReq is *not* delivered to the responder; instead, the driver automatically retransmits the response datagrams that were saved in the list. In this way, the responder never sees the retransmitted TReqs and the requested operation is performed only once.

ATP must include a mechanism for eventually removing XO entries from the responding end's transaction list; two provisions are made for this. When the requester's .ATP driver has received all the TResp datagrams of a particular transaction, it sends a datagram known as a transaction release (TRel); this tells the responder's .ATP driver to remove the transaction from the list. However, the TRel could be lost in the network (or the responding end may die, and so on), leaving the entry in the list forever. To account for this situation, the responder's .ATP driver maintains a release timer for each transaction. If this timer expires and no activity has occurred for the transaction, its entry is removed from the transaction list.