

RETDAT/SETDAT Support

ACReq Task Flow

Tue, Apr 18, 2000

Introduction

This note is an introduction to the IRM support for the Acnet device data request and setting protocols. Some side trips may help serve as background to explain how IRM software supports RETDAT and SETDAT protocols. An earlier document describing the support for these protocols, written in 1990, is called Acnet Data Requests/Settings; it provides an interesting earlier perspective that may include additional detail.

Overall structural flow

The ACReq module in the System code operates as a task that waits on a message queue called ACRQ. The message queue contains reference messages about received Acnet messages, and it is written by the Acnet task when processing a datagram received via the Acnet UDP port number. A reference message merely points to the received Acnet message as it resides in the circular datagram receive buffer. (Concatenated Acnet header-based messages may be used within an Acnet datagram to improve network processing efficiency.) All Acnet protocols share the same message queue so that all such messages pass to the ACReq task for processing in order of reception, independent of the protocol. (Preservation of processing order allows proper support for a SETDAT message that is immediately followed by a RETDAT request to read the most recent setting value.)

The ACReq task supports the RETDAT and SETDAT protocols directly, but it dispatches support for any other Acnet protocol by invoking the appropriate local application that supports it. (It knows which LA to invoke, because each registers its support via the PROTO table, each entry of which includes the Acnet task name and a pointer to the LA table entry, which leads to the LA name and thereby its execution address. A pointer to the reference message that ACReq received is passed to the LA via two of its parameter words.) Whether the message is directly handled by ACReq or indirectly via an LA handler, upon completion of processing the message, ACReq loops back to again read from its ACRQ message queue. This infinite loop comprises ACReq's overall flow.

NetConnect and PROTO tables

Here is an example of the contents of the NetConnect and PROTO tables as taken from node0576, a test node configured similarly to a typical Linac node.

```
FILE<0576>                03/31/00 1117
 0576:00105280    4E43 0130 0180 0011 0000 0000 0000 0000  NetConnect table
header
      taskname  queue-id  task-id  evnt  cntr
:00105290    0011 1A90 0002 04C8 0000 0000 0000 0000  Classic UDP server
:001052A0    414C 524D 0002 04EA 0002 CB1E 0010 0000  D0 alarms server
:001052B0    2108 9A72 0002 0550 0002 CE98 0010 0003  Acnet ALARMR client
:001052C0    4E45 544D 0002 0572 0002 CE98 0010 0000  Acnet NETM client
:001052D0    0011 0007 0002 0594 0002 CE98 0010 0000  UDP Echo server
:001052E0    0011 5506 0002 05B6 0002 CE98 0010 0000  UDP DNS client
:001052F0    0011 5507 0002 05D8 0002 CE98 0010 0000  UDP SLOG client
:00105300    0011 0045 0002 05FA 0002 CE98 0010 0000  UDP TFTP server
:00105310    0011 1106 0002 061C 0002 CE98 0010 0000  UDP DBDL server
:00105320    0011 C3AA 0002 063E 0002 CE98 0010 0000  Multicast events
server
:00105330    0011 1A91 0002 0660 0000 0000 0000 0000  Acnet UDP server
```

:00105340	5250	5952	0002	0682	0000	0000	0000	0000	D0 request server
:00105350	5C71	3C19	0002	06A4	0000	0000	0000	B01E	Acnet RETDAT server
:00105360	9C77	3C19	0002	06A4	0000	0000	0000	0000	Acnet SETDAT server
:00105370	C606	A009	0002	06A4	0000	0000	0000	0B3A	Acnet ACNAUX server
:00105380	B028	7651	0002	06A4	0000	0000	0000	0867	Acnet FTPMAN server
:00105390	5971	A653	0002	06A4	0000	0000	0000	79C1	Acnet REQMON client
:001053A0	0011	5512	0000	0000	0002	CC82	0010	0000	(Ping test page client)
:001053B0	4163	5271	0000	0000	0002	CC82	0010	0000	(ACRQ test page client)
:001053C0	0000	0000	0000	0000	0000	0000	0000	0000	spare
:001053D0	0000	0000	0000	0000	0000	0000	0000	0000	spare
:001053E0	0000	0000	0000	0000	0000	0000	0000	0000	spare
:001053F0	0000	0000	0000	0000	0000	0000	0000	0000	spare

FILE<0576>		03/31/00	1135						PROTO table
	<i>mbTy</i>	<i>offs</i>	<i>Update-ptr</i>	<i>taskname</i>	<i>LATBL-ptr</i>				
0576:00002800	0040	0008	000B	FA5E	C606	A009	0010	7828	Acnet ACNAUX server
:00002810	0011	0008	000B	C9D6	B028	7651	0010	7868	Acnet FTPMAN server
:00002820	0070	0008	0000	0000	0011	0007	0010	78E8	UDP Echo server
:00002830	0077	0008	0000	0000	0011	5506	0010	7928	DNS client
:00002840	007C	0008	0000	0000	0011	5507	0010	7968	SLOG client
:00002850	0075	0008	0000	0000	0011	0045	0010	79A8	TFTP server
:00002860	007D	0008	0000	0000	0011	1106	0010	7B08	DBDL server
:00002870	007E	0008	0000	0000	0011	C3AA	0010	7BC8	ACLK server

The NetConnect table houses both Acnet task connections and UDP port connections, since both share similar dispatching logic. A UDP port connection uses the second word of the task name field for the port number. An entry is in use if the queue-id field is nonzero. In the above example, two entries that were used prior to this sampling are no longer in use. Many entries refer to the same task-id, which in this case is the Update task-id, as their support is provided via local applications. All Acnet protocols use the same queue-id, so that they are directed via the ACRQ task for dispatching. The number of entries available in the NetConnect table is 23.

The PROTO table also holds entries for both Acnet protocols and UDP port protocols. The LATBL-ptr makes the connection to a local application instance and its associated set of parameters, so that the LA can be invoked upon reception of a message in that entry's protocol. The Update-ptr is the address of an Update routine that is called when the System is processing the linked list of active requests. When periodic support is provided for an FTPMAN request, say, it provides the link-up with the FTPMAN-specific code for updating a new reply. The number of entries available in the PROTO table is 16.

The contents of these tables show that support is present for the usual Acnet protocols: ACNAUX, RETDAT, SETDAT, ALARMR, and FTPMAN. Two protocols developed for the D0 detector that were used during Run I are also supported, but it is expected that they will be retired when Run II commences, as the new D0 front ends will be using EPICS.

Several UDP-based protocols are supported. Those that relate to Acnet are SLOG, which logs device settings that are initiated from a front end, and DBDL, which supports automatic downloading into the front end nonvolatile memory structures following DABEL database entry. The latter mechanism assures that central device database changes are reflected in the

front end's own database as far as possible. The multicast clock events protocol is another Acnet-related protocol under study.

Non-Acnet UDP protocols supported are Echo, which is the standard port 7 test protocol, and TFTP, which allows downloading of local and page applications into the nonvolatile memory file system of front ends. Client support is provided for Domain Name Service queries, used to build mappings between front end node numbers and IP addresses. Client support for NTP, the network time protocol, is used to keep up with calendar time-of-day. One or two nodes acquire this information every few minutes and share it with all other front ends.

NETM and REQMON are dummy protocols used to monitor the health of network activity. NETM is used only in token ring front ends to insure that their token ring receiver hardware is functional. It will be retired once token ring is no longer in use. REQMON is used to verify that clients that have active requests of whatever protocol remain accessible. After continued failure to elicit communications from a requesting node, its requests are canceled.

ACReq initialization

During initialization of the ACReq task, the ACRQ message queue is created, and both RETDAT and SETDAT are registered in the NetConnect table so that any message of either protocol is written into the same ACRQ message queue. (LAs that support other Acnet protocols also use the same message queue. As a part of its initialization, such an LA obtains the ACRQ message queue id for this purpose.)

If a RETDLOG data stream is defined in the DSTRM table, its index# is saved for use as a diagnostic log that includes an entry for each RETDAT request, including cancels, seen by ACReq task. The information recorded includes the requesting node, the expected reply length, the number of device packets, the FTD, the message id, and the time-of-day.

After task initialization, the main loop that is described above is entered to process all Acnet messages passed to it via its ACRQ message queue.

Message processing

When a message is received and passed to the ACRQ message queue, and the ACReq task is awakened, NetCheck is used to learn of it. The reference message obtained from the message queue includes the requesting node, a pointer to the Acnet header that immediately precedes the message itself, and the message size, including the Acnet header. If the indicated task name is neither RETDAT nor SETDAT, then the PROTO table is searched for an LA handler that can process it. If the message is a cancel for an existing RETDAT request, then that request is canceled. If the Acnet message type is a request, then either REQUEST or SETTING is called to process it. If it is a reply message type, SREPLY is called to process either an answer fragment or setting acknowledgment from another node as a result of providing server support for a request. Upon return from any of these routines, any resulting error code is returned in an Acnet status-only reply, assuming the message type of the original message was a request.

RETDAT message format

As a review, the message format used for a RETDAT message consists of three-word header followed by an array of 16-byte device packets. The three words specify the expected reply message size, the number of device packets, and the frequency-time descriptor, or FTD. The FTD, if positive, specifies the reply period in units of 60Hz, so that 15Hz would be indicated by a value of 4. An FTD with the sign bit set means that replies should be returned upon the occurrence of the clock event specified in the low byte. An FTD with the value zero implies a one-shot request. A device packet includes a 4-byte structure of the 1-byte Acnet

property index and the 3-byte Acnet database device index. The next 4 words are the SSDN, described in the next section. The last two words are the length and offset words, to be described later.

The reply message format used by RETDAT uses no special header, but is merely a series of structures, each of which is a status word followed by the requested device data. As an example, a request for a 2-byte analog reading and a 20-byte analog alarm block would result in a reply message that consists of a status word, a 2-byte reading, a status word, and a 20-byte analog alarm block, 26 bytes in all.

SETDAT message format

A SETDAT message is made up of a two-word header followed by a sequence of device setting structures, each of which is a device packet followed by its associated setting data. The setting data is padded, if necessary, to an even number of bytes. The header is a flag word and the number of setting device packets. The least significant bit of the flag word, if set, means that the setting should be forwarded to the central database. The device packet structure is the same as that described above for RETDAT. For the simplest case of a two-byte setting to a single analog device, without forwarding, the SETDAT message would consist of the words 0, 1, an 8-word device packet, and one word of data.

The reply to a SETDAT request-type message is an array of status words, one for each device setting packet that was used in the SETDAT request message. In the above simple example, the reply message would consist of a single status word.

SSDN format

The format of SSDNs used by the RETDAT/SETDAT support is fairly restricted. The layout for an SSDN is as follows:

<i>Word#</i>	<i>Meaning</i>
1	Listype# in high byte, flags and ident length in low byte
2	Node# that actually sources this data
3	Ident (channel#, bit#, other index#)
4	Ident second word if needed, else data item size in lo byte.

Two non-English words are used to describe this formulation. The *listype* number in the high byte of Word 1 specifies the type of data being accessed. The *ident* is a specification of which instance of that type is addressed by this request. It is most often a table index# whose actual meaning depends on the listype# in the high byte of Word 1. Typical examples refer to fields in records of tables for which the ident specifies the table entry#. Use 00 for analog readings, 01 for D/A settings, 02 for alarm blocks, all of which reference a specific analog channel# via the ident word. In object-oriented jargon, a listype number may be a class id; an ident may be an object id that identifies a specific object in that class.

Support for listypes and idents is fundamental to IRM software support of data access of all kinds, which is what prompted coining these abstract terms. It was originally used for the Classic protocol. Each listype implies a particular ident format, always an even number of bytes in length ≥ 4 . The first two bytes are the node number word, which is the Acnet trunk number in the high byte and the Acnet node number in the low byte. The maximum ident length in use today is 14, including the 2-byte node number, which allows access to a nonvolatile memory-based file by specifying an 8-character file name and a 4-byte offset. Only idents up to 6 bytes in length, can be specified in the 8-byte SSDN. The low 4 bits of Word 1 give the ident length; only two values are valid. A value of 1 implies a 4-byte ident is specified in Words 2 and 3; a value of 2 implies a 6-byte ident is specified in Words 2, 3, and 4.

Length and offset

The content of an SSDN for an Acnet device/property comes from the Acnet central database and is therefore fixed, but a length word and an offset word may be specified by the user application program. Both words are given special support for RETDAT and SETDAT requests. The flags in the upper 4 bits of the low byte of Word 1 specify how the offset word is used. If this nibble is 0, it means that the offset carries its original meaning, as an offset into the data structure accessed by the request. There is no support for this generic use of the offset word, so in this case, if the offset word is nonzero, an error is returned to indicate that no such support is available.

If the flag nibble is 1, the offset is used to modify the non-node part of the 4-byte ident specified in Word 3, or the 6-byte ident specified in Words 3–4 treated as a long word, by simple addition. This permits using a single Acnet device to access an arbitrary entry in a System table, each of which is an array of fixed-size structs. In such cases, the value of Word 3 is usually zero, so that the value of the offset is merely the table entry number. For 6-byte idents, this scheme can be used to access up to 64K of memory beginning at any byte address. (Several listypes are used to provide access to arbitrary memory addresses.) It has also been used for accessing 1000Hz digitizer data, in which a two-word ident in Words 3–4 uses an analog channel number in the first word and a clock event number in the second. The application program can thus specify the event number to be used as the reset event for the returned time stamp values.

If the flag is 2, the offset word is used to modify the two-word ident in Words 3 and 4, by addition of specified offset*256 to the long word value. This is designed to allow a single Acnet device to provide access to memory in a 16MB range.

Besides the offset word, optional support is also included for handling the length word, which is the number of bytes requested for an Acnet device. This support allows access to array data as referenced by consecutive idents. For example, one Acnet device may refer to a given analog channel reading. A request specifying a length of 2 returns that channel's 2-byte reading. A request for 30 bytes, if this option is enabled, returns an array of 15 readings of consecutively-numbered analog channels. The option is only available for 4-byte idents, so that Word 4 (low byte) can be used to specify the size of one item, which for this example would be 2. The number of bytes requested must be an exact multiple of the item size value. If the item size is zero, the option is not enabled, and such a request for 30 bytes would merely return the 30 bytes of memory beginning at the two bytes of the indicated analog channel's reading, unlikely to be interest for most users. One would enable this option only in special cases for which it makes sense. The special support for both offset and length can be enabled for the same SSDN.

Server/consolidator logic

Given the above SSDN format, with the node number always specified in Word 2, it can be seen how the server/consolidator logic works. Consider a RETDAT request that includes a number of devices. For each device, the Acnet property index, the length, the offset, and the SSDN are specified. The REQUEST handler routine calls PREVIEW to determine how it shall be handled. For the simplest case, in which all device packets include SSDNs that reference the local node, the request is handled directly by the receiving node as a non-server request. If all SSDNs specify a single outside node, the request is forwarded to that node for processing, after first preparing support for the replies that will subsequently be returned to the requesting node. If there are at least two different nodes represented in the SSDNs of the device packets, the same server treatment is used, but the request is forwarded to a multicast destination to reach the other nodes involved. (Each node is configured with a "broadcast" node# that is to be used for these cases. By assigning different multicast addresses for different projects, we keep such

forwarded requests within a single project. Linac server nodes, for example, only forward such requests to Linac nodes.)

The careful reader may perhaps wonder why there is no danger of a "chain reaction" of forwarding activity. Besides the node numbers indicated in the SSDNs, the question of whether the request was received via multicast is part of the server/nonsERVER decision process. If a request is received *via multicast*, it is handled as a nonsERVER request, and any outside node devices included in the request are ignored. (If that leaves no local node references left, the request is ignored completely.) In this way, a server node may receive reply fragments from multiple nodes in response to a multicast forwarded request. Each fragment is part of the complete reply message to be returned to the requester. The SREPLY routine processes these replies, carefully copying them into the complete reply structure for later delivery by the Server task.

REQUEST handling

This routine examines both server and nonsERVER requests. It calls PREVIEW to collect the statistics required to determine whether the request will be treated as a server or nonsERVER request. The statistics gathered by PREVIEW are as follows:

- multicast flag
- #packets from local node
- first non-local node number
- #packets from first non-local node
- error status, including null case (multicast and no local packets)

The routine DOREQ is called to allocate and initialize the request block that will be used to support the request while it is active. For the server case, a new message id is obtained for use with the forwarded request. Then either SERVER or NSERVER is invoked to perform the major support of completing request initialization.

Certain limits are checked to insure that the request is reasonable. These limits are 300 device packets, 1200 idents, 16K reply size, and the maximum number of bytes of reply specified in the request header. The 16K limit comes from the maximum datagram size supported via UDP/IP. The number of idents can be larger than the number of device packets when an Acnet device refers to a sequence of analog channel numbers.

For the nonsERVER case, an internal pointers block is allocated and filled to support the subsequent updating of the request. Every data request is processed in two stages. The first stage initializes and prepares the request for the second stage, which is the reply updating stage. If the original request is analogous to program source code, the internal pointers are analogous to object code. The initial processing of a data request is analogous to compiling source code and results in an array of internal pointers that are then processed in order to generate the reply data. This approach to request handling is done for efficiency during reply updating, as replies may be generated as often as 15Hz. A one-shot request is handled the same way. It is first initialized and prepared for updating, then immediately processed to generate the reply update. Once the reply has been delivered, a one-shot request is canceled, and all support structures are released.

For the server case, no internal pointer block is used, but an external request block is allocated to support the forwarded request. The "object code" for a server case, included in the request block, is the array of references to the position in the reply buffer for each of the device data packets from the contributing nodes. When a reply fragment is received from a contributing node, each of its device data contributions is copied to the appropriate position in the eventual

reply message as specified by the array of references.

Request data structures

Multiple request structures are allocated to support a RETDAT request. The first is the request block, which contains references to other structures and the original 3-word request header. Three other structures may be referenced by the request block. The internal pointers block contains the array of internal pointers for use in building replies. An external request block contains the copy of the original request message forwarded when providing server support for a request. The forwarded request is the same as the originally-received request message, but its Acnet header is modified to use a different message-id as well as source and destination nodes. The reply message block, the third block referenced by the request block, contains the reply message for eventual return to the requester. The reply block also includes a pointer back to the request block. So, support for a RETDAT request involves two additional allocated structures for the request. The external request block that is used to support a server request is retained until the request is canceled, because it may be needed to reissue the original request to a contributing node that does not respond. Any such reissuing is always sent to an individual node; it is not multicast. If many nodes who must contribute to a request refuse to respond, then many reissued requests will be sent. They will continue to be sent every two seconds as long as the request is active; i.e., it has not yet been canceled. Such reissues are not attempted for one-shot requests.

An immediate reply is always generated for a periodic request, just as for the one-shot case. The exception is a request that is to be returned upon a clock event, in which no reply is generated without first sensing an occurrence of that clock event following request initialization. The server must have the same clock event information as the contributing nodes, so the server can decide on which cycle to deliver the reply.

The first reply to a nonserver request is delivered immediately after request initialization, except for the case of an event-based request. Subsequent replies are delivered as soon as the data pool has been updated, and the reply 15Hz period counter indicates that a reply is due. This normally occurs quite early in a 15Hz cycle, after updating the data pool.

Replies to server requests are normally delivered at 40 ms past the start of the front end node's 15Hz cycle activity, which commences after an interrupt that occurs at a certain time delay after a 15Hz clock event. An exception is made for the first reply to a one-shot server request, in which the composite reply is delivered to the requester as soon as the first reply fragment has been processed from all contributing nodes. This also provides prompt replies to one-shot requests.

The time delay after a 15Hz clock event is chosen to be the same for all nodes in a project, so that all nodes operate synchronously. In the Linac project, for example, the delay is 3 ms after a 15Hz clock event. The Linac beam is accelerated at 2 ms after the event, so that the Linac nodes receive their 15Hz interrupt at 1 ms after the Linac beam pulse. This is long enough for the Linac RF pulse to be off as well as the pulsed quadrupole power supplies, so that the Linac gallery is relatively quiet. Most Linac signals are outputs of sample-and-hold circuits that are triggered at a time during the 40 μ s Linac beam pulse.

The time delay for the Booster HLRF nodes is set for 35 ms, which is at Booster beam extraction, which is 33 ms after Booster beam injection from the Linac. At this time in the 15Hz, any Booster signals of interest will have occurred.

It is important that all nodes in a project use the same delay following a 15Hz event to

synchronize their operation, lest the server support for data requests that span multiple nodes fail to return correlated (at the 15Hz level) data. Correlation between Booster and Linac data, since they are from different projects and operate at different times within the 15Hz cycle, is expected to be handled at a higher level. Such correlation means that the data was measured on beam from the same 15Hz cycle.

Data request handling

Each active data request is supported by a request block of a type that varies according to the request protocol. A linked list of active request blocks is maintained in an order that groups together multiple requests of the same protocol whose replies target the same requester. The time to update and deliver replies for all active nonserver requests is early in the cycle just after the data pool is updated for that cycle. For each request in the linked list, the appropriate protocol-handler is invoked to build a reply if one is due on that cycle, and to queue that reply message to the network. At the end of the linked list, the network queue is flushed, which causes all queued messages to be combined into datagrams and delivered to the appropriate target nodes. The order in which the linked list is maintained increases the likelihood that consecutive messages will be found in the network queue that are destined for the same node, so that they are eligible to be combined into a common datagram. In some cases, this can make a large difference in network processing efficiency.

Server data requests are updated at a later time within the 15Hz cycle, generally at 40 ms. This delay gives time for the server node to receive replies from the all contributing nodes of a request before a reply is delivered to the original requester. The linked list is traversed, and replies are delivered for all server requests that are due on that cycle. The server node monitors the replies from the contributing nodes; if any has not responded in a timely fashion according to the specified reply period of the request, the reply status word for the device data from that node is marked with 36□7 error status, or in the case that the node has never responded to the request at all, 36□8 error status. As long as the request is active, the server node reissues the request to such a non-replying contributing node every two seconds in hopes that it will eventually join in contributing its data to the request. The server node, of course, knows which nodes should respond to a given request, because it analyzed the SSDNs of all device packets in the request.

Note that having synchronous operation across the front ends of a project is important for server support to work. The server node knows when to anticipate receiving replies from the contributing nodes, because it operates synchronously, too.

When a RETDAT request message is received that requires server node support, but includes devices that belong to the server node itself, the forwarded multicast request must be receivable by the server node. (IRMs achieve this by configuring the ethernet interface on the MVME-162 board in a special mode so that it receives all transmitted frames that are logically addressed to it.) When it receives the forwarded request message, it reacts to it just as if it had been sent by any other node, arranging to return replies to itself. It acts both as a server node and a nonserver node for the same request. This was done so that its own contribution to such a request did not require any special handling.

Data averaging

In order to support Linac data at 15Hz with Acnet requests of 1Hz, analog readings are averaged in a special way to produce the values included in a request for such data. The special way means that beam cycles are preferentially averaged. Only if no beam cycles occurred during the 15 cycles over which the average is computed will the reply be the average of non-beam cycles; otherwise, it is the average of the beam cycle readings only. Note that if a given

node has no beam status available, so that the dedicated Bit number that carries that status is fixed, at either 1 or 0, the average is the simple average over the number of cycles involved. The averaging logic for analog readings is enabled for all requests with reply periods more than one cycle.

Setting handling

The SETDAT protocol supports setting messages that can be sent either as Unsolicited Messages (USMs), in which no acknowledgment can result, and Request messages (REQ), in which an acknowledgment must be returned. An acknowledgment includes an array of status words, each resulting from a setting action. Each status word is in the usual Acnet format; the high byte is an error status value, often negative, and the low byte is a facility number. The facility code assigned for Linac/IRM nodes is 36. A zero status word (both bytes) means there was no error.

For server support to a setting request message, the server node initializes a setting acknowledgment block to support the communications. This is analogous to a data request in which the reply data includes only status words. Setting acknowledgment replies from the various contributing nodes are combined in the same way that reply fragments are combined for server support of a data request. As soon as the last status fragment of a server setting request is received, the combined status reply is delivered to the requesting node.

Reply updating

When processing new requests, an event is sent to the Update task to cause all new requests to be updated. This produces a prompt first reply to a periodic request, or the only reply to a one-shot request. Multiple new requests may be queued before the request-processing task completes, so that updating all new requests may update several requests. Again, since the linked list of active requests is always maintained in an order that groups requests targeting the same node, processing replies for multiple requests may likely result in multiple replies sharing the same datagram, thus increasing network handling efficiency.

During QMonitor task processing, which occurs at least every cycle, monitoring the messages that have been completely transmitted, any reply message of any Acnet protocol causes ACDELCHK to be invoked, and if the request was a one-shot request, the request is canceled and all its associated data structures are freed. As stated before, event-based requests do not produce immediate replies; every such reply must await the occurrence of the clock event.

During Update task processing, while traversing the linked list of active requests, any nonserver Acnet RETDAT request results in a call to ACUPDCHK, and if it is time to build a reply, it does so and queues the reply message to the network. As an extra step in keeping network communications flowing, just before queuing a reply to the network, it checks whether the message at the top of the queue is one of the same general type that targets the same node. If it is, the queue is flushed to the network before the new reply is placed onto the queue. In this way, as replies switch to new target nodes while traversing the linked list of active requests, they are promptly delivered to the network hardware.

During Server task processing, which normally occurs at 40 ms after the start of the 15Hz cycle, the linked list of active requests is again traversed, and ACUPSERV is called to update and queue to the network replies to all server requests that are due. For server requests, the determination of when replies are due is understood simultaneously by both the server node and the contributing nodes. If a request specified 1Hz replies, for example, they are delivered by the contributing nodes during the early part of the same cycle on which the server node delivers the aggregate reply to the requesting node. This is another reason why all nodes in a

project need to operate synchronously.

When a contributing node to a server request fails to return a reply, it is periodically reminded of the request in hopes that it will "come alive" soon. To reissue the request to a single node, when the original request was multicast, requires some care. If such a request is resent directly to a contributing node, that node will determine that it should be a server to the request, which is *not* what we want. (The reason the node will think that it should be a server is because it did not receive the request via multicast.) In order to achieve what we want, all the device packets associated with that node are extracted from the original request, and only the abbreviated request is sent to the contributing node. Its subsequent reply will look the same as it would have when receiving the entire request via multicast, because contributing nodes do not include "holes" in their replies; they reply as if the multicast message they received contained only their own devices.

Low level interface

The RETDAT and SETDAT logic pertaining to control system devices rests above a low level interface that was designed along with the original Classic data request protocol. The made-up terms listype and ident were described above. During request initialization, idents are converted into internal pointers, which are then processed during update time to produce an updated set of answers that comprise the reply message. A RETDAT request includes an array of device packets, each of which includes an SSDN that carries the listype and ident values referenced by the Acnet device and property.

To initialize an Acnet RETDAT request, each device packet is handed to the appropriate pointer-type routine that generates an internal pointer, or in the case of an array device, an array of internal pointers. The pointer-type routine used depends upon the listype. During the processing of each Acnet device packet, many error conditions are checked; if any is found, an Acnet status-only reply is returned to the requester. Assuming that no errors are detected during request initialization, the request is formally accepted as active.

When it is time to produce a reply, each Acnet device packet is again processed, but this time, by using the internal pointers generated at initialization to produce a status word and reply data for each packet. This is done via a read-type routine that depends on the listype number. The internal pointer is of a format that the pointer-type routine can produce and which is interpreted by the read-type routine to produce reply data. The ident information is only used during request initialization; for updating replies, only the internal pointer is needed. The information gleaned from each Acnet device packet also includes the number of bytes requested, the number of internal pointers (> 1 only for the array case), the read-type routine, and a post-processing routine. This information enables the read-routine to produce reply data for an Acnet device packet.

For server request processing, the information saved for each Acnet device packet is somewhat different. It includes the number of bytes requested, the target node number, the offset into the reply buffer, and the age of the latest reply fragment. This information is processed upon receipt of a reply fragment from a contributing node to guide the proper placement of the reply status and data into the complete reply message. Monitoring the age of the latest replies allows detecting when replies are tardy, which may result in sending a reminder request to that node in hopes that it will join the rest of the contributing nodes in building reply data. The reply data from each contributing node includes only the device data from that node. But the server knows which nodes are expected to contribute to which device packet, so it can copy the pieces into the reply buffer where they belong. When the deadline time for server requests is reached, the reply buffer is already prepared and ready to be queued to the network.

There is a listype table that holds certain parameters for each listype. These parameters are as follows:

- ident-type number
- read-type number
- set-type number
- maximum number of bytes of settable data
- pointer-type number, or System table number

A System table number is used for those listypes that provide simple access to a field of a System table. The ident-type number implies an ident length (always even) that is currently in the range of 4–14 bytes. Listype numbers currently range from 0–87.

Error status returns

Many error status returns are possible when initializing a request, and each results in an Acnet status-only reply message to announce failure to process that request. After a request has been initialized, and no errors were detected, there is the chance that error status could be subsequently returned with the reply data. Only three error return status values are possible with the reply data. From a nonserver node, only bus error status can be returned, which is 36□4. A server node that receives this from a contributing node will of course pass it on in its complete reply to the requester.

For a server periodic request, two other error codes may result, each relating to the responsiveness of the contributing node. If no reply fragment has been received from a contributing node since the request was initialized, a 36□8 is returned. If a reply is missing from a contributing node, but has been received at least once since the request was initialized, a 36□7 is returned. If a node is down, one can expect the former; if a node goes down after request initialization, or if a node is operating asynchronously or at a slower rate than the server, or if there are network problems, the 36□7 code may occur. (Note that the tardy error status code is not returned for event-based requests.) All other error codes are filtered out during request initialization.

The number of potential error returns from SETDAT are many. When processing a SETDAT message, each indicated setting is performed in turn. The set-type routine returns an error code that is saved in the array of status words to be returned in the setting reply message. Server support is provided for settings. But settings are processed in one pass, unlike RETDAT. If the PREVIEW routine returns multicast status, or if the number of local packets is equal to the total number of packets, nonserver processing is performed; otherwise, server processing is performed, for which the setting message is forwarded to either the "broadcast" (multicast) destination, or to a single target node if the number of the first non-local packets is equal to the total number of packets.

RETDAT Timing

Having described the processing supported by IRMs for the RETDAT protocol, a few timing measurements may be helpful to bring things into proper perspective. This section includes some of these measurements

The following timing measurements are made from task timing diagnostics, with all times given in ms. The meanings of the column headings are as follows:

SNAP	The SNAP task performs IP network support upon datagram arrival
ANet	Acnet task dispatches message based upon task name or task-id
ACReq	RETDAT processing, either for requests or replies
Updt	Update task processing builds initial reply
Total	Sum of timing for SNAP, ANet, ACReq, Updt
Δ Updt	Approximate time for Update task to build reply
Δ Serv	Approximate time for Server task to build reply

NonsERVER request timing

The following statistics measure the time for an IRM to handle a **nonsERVER** request. Several sizes of requests are used, with varying numbers of devices. Each one consists of a request for a 2-byte reading and a 2-byte setting for each device, or two PI/DI's per device. The requests are for 7.5Hz replies. This allows measuring the time for readings averaging that occurs every cycle, whether the reply is to be delivered or not.

<i>#dev</i>	<i>SNAP</i>	<i>ANet</i>	<i>ACReq</i>	<i>Updt</i>	<i>Total</i>	<i>ΔUpdt</i>
1	0.19	0.10	0.35	0.42	1.06	0.15–0.3
10	0.22	0.10	0.81	0.62	1.75	0.2–0.4
20	0.26	0.09	1.31	0.80	2.46	0.4–0.9
40	0.34	0.10	2.35	1.19	3.98	0.1–1.0
80	1.10	0.10	4.45	1.98	7.63	0.3–2.0

The time needed to provide IP support depends upon the size of the message, since the UDP checksum must be checked. The times needed by the Acnet task are constant, since Acnet only does dispatching, and the message does not have to be copied anywhere. The RETDAT time (ACReq) depends upon the number of devices, as each device packet is processed as the request is initialized. The Update task time is used to build the first reply and pass it to the network hardware. The last column shows the ongoing extra time needed in the Update task to build a reply each time it is due. The first number is the time when a reply is not due, in which a (short) time is needed to accumulate readings for computing an average value. The second number is the time to actually build a reply, which is of course similar to the time for building the first reply. The times in the Updt column and the Total column summarize this data. For example, the time needed to fully process reception of a 20-device (40 packets) nonsERVER request is about 2.5 ms, with about 1 ms needed to build each reply.

Here follows network activity related to the 20 device case, which uses 40 device packets. The request message is received and initialized, and a reply is built for a prompt first response. All device packets are local, so this is a very simple case. The turnaround time is 3 ms. Subsequent replies are delivered every two cycles at a time early in the 15Hz cycle.

```

Node Size  Ptr   HrMn Sc Cy ms
E0D1 02B0 R 163222 1514:04-05+34 Request
E0D1 00CA T 18CD4A 1514:04-05+37 First reply to requester
E0D1 00CA T 18CE2E 1514:04-07+ 2 Second reply
E0D1 00CA T 18CF12 1514:04-09+ 2 Third reply

```

SERVER request timing

Here are times for a **SERVER** RETDAT request for 2-byte analog readings and settings to be delivered at 7.5Hz. To make this a simple example, all device packets refer to the same node that is not the node receiving the request.

#dev	SNAP	ANet	ACReq	Updt	Total	SNAP	ANet	ACReq	Updt	Δ Serv
1	0.19	0.09	0.54	0.29	1.11	0.18	0.08	0.20	0.28	0.1
10	0.21	0.10	0.63	0.34	1.28	0.18	0.09	0.26	0.30	0.13
20	0.30	0.10	0.73	0.44	1.57	0.19	0.08	0.35	0.32	0.14
40	0.33	0.10	0.95	0.54	1.92	0.21	0.09	0.51	0.34	0.20
80	1.55	0.10	1.40	1.09	4.14	0.26	0.09	0.81	0.44	0.31

There are more steps for the server case, because the data must be retrieved from another node. The RETDAT time (ACReq) depends somewhat on the size of the request, but it only has to forward the request, as the request included none of its own devices in this test example. In summary, the time to initialize the server request for the case of 20 devices (40 packets) is about 1.6ms. The time to process each reply from the contributing node and build the reply message for the requester is about 1.25 ms. These times are somewhat smaller than those for the nonserver case, as the server node is not doing the bulk of the work of actually generating the data.

Here follows the network activity for the 20-device (40 device packets) server case. For this simple test, only a single contributing node was used. When the request was received, it was forwarded to the other node, which initialized the request and delivered a prompt first reply to the server node, which in turn passed it on to the requester. The total turn-around at the server node for the first reply was 6ms. Subsequent replies are delivered at 40 ms into the IRM's 15Hz cycle, allowing plenty of time to receive all reply fragments.

```

Node Size  Ptr      HrMn Sc Cy ms
E0D1 02B0 R 16B622 1445:01-07+46 Request
E131 02B0 T 18EA6C 1445:01-07+47 Forward request
E131 00CA R 16BC22 1445:01-07+51 First reply from (only) contributing node
E0D1 00CA T 18ED36 1445:01-07+52 to requester
E131 00CA R 16C222 1445:01-09+ 3 Second reply from contributing node
E0D1 00CA T 18EE1A 1445:01-09+40 to requester

```

Here follows the network activity related to the 80-device (160 device packets) case. Because of the large size of the request message datagram, it arrives in fragments which first have to be reassembled before the message can be formally received for processing. For this test, the requesting node was a test front end on token ring, which uses a 2KB MTU; this is why the request arrived in 3 fragments but was forwarded on ethernet in only two fragments. Because the reply message size is only about 700 bytes, no fragmentation is used for its delivery. In this example, the contributing node received the request late in the cycle, so its reply was slightly delayed due to updating its own data pool early in its cycle. Even so, the turn-around time at the server node was 20ms as measured from the arrival time of the first request fragment to the delivery of the first reply.

```

Node Size  Ptr      HrMn Sc Cy ms
E0D0 0248 R 163822 1333:21-02+54 Fragment #1
E0D0 05D8 R 163E22 1333:21-02+57 #2
E0D0 0230 R 164422 1333:21-02+57 #3
E0D1 0A30 R 031854 1333:21-02+58 Accept reassembled datagram
E131 05D8 T 18335C 1333:21-02+60 Fragment #1 (Forward entire request)
E131 0468 T 183DA6 1333:21-02+61 #2
E131 02AA R 164A22 1333:21-03+ 6 First reply from contributing node

```

```
E0D1 02AA T 184228 1333:21-03+ 8 to requester
E131 02AA R 165622 1333:21-05+ 5 Second reply from contributing node
E0D1 02AA T 1844EC 1333:21-05+41 to requester
E131 02AA R 166222 1333:21-07+ 5 Third reply from contributing node
E0D1 02AA T 1847B0 1333:21-07+41 to requester
E131 02AA R 166E22 1333:21-09+ 5 etc.
```

More complex examples of RETDAT timing can easily be measured, but the above should provide a picture of the real time operation of the IRM front end support for the RETDAT protocol that is described in this note.