

# Node Number Notes

*An analysis*

Tue, Feb 6, 2001

This note is an attempt to analyze some of the thinking behind the node number formats used internally within the system software. The focus is on the use of the “ethernet bit,” or E-bit.

## *History*

In the beginning, which is to say around 1980, there was only a single network supported by the system software. That network was token ring, and Internet Protocols were not yet supported. Node number values in use were of the form `0x0xxx`, with `0x00Fx` used for multicast, or group addressing, as it was called for token ring. The relationship of the node number to the 6-byte physical network address presumed local administration of the network addresses. A fixed 4-byte prefix is used, with the 16-bit node number used for the final 2 bytes.

In order to interface properly with unix computers, it was decided to support the suite of Internet Protocols, since unix networking assumed the use of UDP/IP. But the socket specification as used for IP support was cumbersome, requiring a 32-bit IP address and a 16-bit UDP port number. All the network related support within the system used a 16-bit node number. It was decided to find a way to abbreviate the socket specification to 16 bits; thus was born the concept of a “pseudo node number.” The P<sub>SN</sub> has the format `0x6xy`, where the `xx` is the index into the IPARP system table, each entry of which includes an IP address, a physical network address, and a pointer to a “Port Number Block,” which includes up to 15 UDP port numbers in use by that node. The `y` value is an index into the list of port numbers within the PNB. The limitations of this scheme are that only about 250 nodes can be dealt with at one time via IP, each of which can have up to 15 active UDP port numbers in use. The `y` value of 0 is used for non-UDP/IP communications, such as ICMP. (TCP is not supported, as it was deemed unsuitable for real-time, synchronous 15 Hz, communications.)

With the upgrade to Linac controls in 1992, it became necessary to support arcnet communications with the Smart Rack Monitors, which are distributed systems connecting to the Linac hardware analog and digital interfaces. There are typically 4–5 SRMs connected via arcnet to a single front end node, so the need for addressibility of arcnet nodes is not extensive. The arcnet interface itself uses an 8-bit value to specify an arcnet node address. A 16-bit node number format of `0x7Axx` is used for the purpose within the system software. Actual SRM node addresses can range from `0xA1–0xBF`, although only values in the range `0xA1–0xAB` have actually been used, with `0x7A00` specifying broadcast addressing, the only multicast addressing available on arcnet.)

The next major upgrade to network support was the addition of ethernet, this time for use with Internet Rack Monitors, each of which is a full-fledged front end node with a 3-slot VME crate within that required only the CPU board slot to be filled. To accommodate ethernet addressing, the most significant bit of the 16-bit node number is used. This allowed for both raw (non-IP) as well as IP to be supported, although raw ethernet was seldom used except during development.

As an additional complication during the entire period, Acnet console software added

support for IP. They also chose a 16-bit node number representation for each front end. These Acnet IP node numbers use the range 0x0900–0x10xx, with values of 0x09Fx allocated for IP multicast addressing. For a node to determine the IP address that corresponds to each such node number, it is necessary to download a table from the Acnet system, and only *bona fide* Acnet nodes may download this table. The value 0x09FF is reserved for IP broadcasting.

The Acnet scheme for IP addressing is in contrast to the scheme for relating tradition node numbers of the form 0x0xxx to IP addresses used by IRM software, in which the Domain Name Server is accessed to obtain the correct IP node address. For this to work, the names registered with the DNS are node0xxx with a fixed suffix such as “.fna1.gov” entered with a field in the IPNAT system table. The local application DNSQ manages this table of node number and IP addresses, re-interrogating the DNS about every 8 hours to keep its contents from growing stale. The IPNAT is in non-volatile memory, so all previous knowledge of such relationships are restored when a front end boots. When the system software attempts to address a node via IP using the native form 0x0xxx, the IPNAT is searched for a match; if none is found, a message queue is used to alert DNSQ to the fact. DNSQ subsequently requests the IP address from the DNS using the standard name convention, and when it receives a proper reply, it places the IP address in the IPNAT entry.

Until the time that the IPNAT is filled by a DNS reply, any IP communications to that target node are ignored, to prevent holding up normal system task communications with other nodes. Once the IP address is known in the IPNAT, however, such a delay will never occur again for any communications by that front end with that target node. Note that this “failure to communicate” only occurs for cases in which the front end initiates communication with another such front end; it does not happen for cases in which the front end is replying to a request initiated from another node, front end or otherwise. The received request message allows access to the requester’s IP address and source UDP port number, so a Psn is used internally, and IPNAT is not needed.

A Psn is used internally for IP communications as a shorthand for a UDP socket. But communications can be initiated from a front end to a target node using its native node number, or using its Acnet IP node number.

In summary, here are the node number ranges in use by the system software:

<i>Node range</i>	<i>Meaning</i>
0500–07FF	Native node#, either raw or IP
0800–08EF	Token ring to Acnet node via ethernet bridge
0900–10EF	Acnet IP node, addressed via Acnet Trunk tables
6020–6FFF	IP target node/port via IPARP table
7A00–7ABF	Arcnet communications with SRMs
8500–87FF	Native node# via ethernet
E020–EFFF	IP target node/port via IPARP table on ethernet

The E-bit is used to tag a node number for ethernet communications. But it is not always required, depending on the hardware support available in a front end. It is of course necessary when referring to raw communications from a node that supports both token

ring and ethernet interfaces. But such nodes have only existed during development. It is more likely that a node has either token ring or ethernet interface available. (At this writing, token ring support is being phased out, so the climate is even simpler.)

When a native node# is used, the use of raw or IP is available, depending upon a configuration value in the third word of the `PAGEM` system table. For modern front ends that use IP communications only, this "broadcast node" value should be of the form `0x09Fx`, which implies IP multicasting, used with the network interface that supports IP. (Only one network interface can support IP in a front end; with IP, one should be enough.) For those token ring nodes configured not to use IP when they target communications, it should have the value `0x00Fx`. When using a native node# for IP communications, the target port# depends upon the context.

### *Message-based communications*

Within the system support for networking, both frame/datagram and messages are used as a unit. It is possible to build a datagram to send to UDP port on a target node. But it is also possible to operate at the message level. Such message-based communications can only occur using either Classic or Acnet message protocols. Classic protocols use UDP port number 6800, and Acnet protocols use UDP port number 6801. The service provided by the system for message-based communications is one of concatenating multiple messages into the same frame/datagram that are targeted for the same node at the same time.

The support of network transmissions in either case is done via an output pointer queue. A message block is allocated, and the message is constructed within the block. the message block is queued to the output pointer queue, or `OUTPQ`. When the `NETXMIT` routine is invoked, consecutive queued entries in the `OUTPQ` are scanned, and successive entries found that target the same node/port are combined into a single datagram for transmission to the target node. (Note that different ports of the same target node are addressed using distinct `PSN`'s, so that multiple ports of a target node are not targeted within the same datagram!)

A separate `OUTPQ` is used for each supported network. When `NETXMIT` is called, its argument is a network number, so that its message-combining logic applies to all networks for message-based communications. Via the network number, `NETXMIT` determines which `OUTPQ` to scan for messages to be combined and queued to the network interface. The basic idea behind much of the organization of all system software is to keep things moving. Network communications to any one node should not unduly hold up network communications to other nodes. Note that this approach is not in keeping with the use of `sendto()` in the standard socket interface, and this problem had to be addressed in the latest PowerPC/VxWorks-based version of the system.

### *Non-networking networking*

The Classic protocol is supported via software that doesn't even imply networking directly. A data request is made for a list of "devices," here used in the most general sense. The specification for each such device in the list is via an "ident," whose format consists of a node number followed by one or more 16-bit words of indexing information. The Classic protocol requester, therefore, has to know the node numbers used for the ident's in the list, but that's all the networking that it has to know. The exception to this might be the device

name lookup that applies to analog channels. A special data request can be made in which the ident is merely the 6-character name, without a node number, the reply data to which includes the node number and channel, comprising a channel ident that may be used in subsequent data requests.

After building a list of idents for which data is sought, a single call conveys the data request to the system software, which takes action to see it is fulfilled. If some of the idents are local to that front end, obtaining the data is contained within the node. If other idents indicate external nodes, then networking is used to acquire the same data from other nodes. When more than one external node is referenced by a single request, then multicasting is used to forward the request for all external idents to all relevant nodes.

After submitting a request, a client application should allow time before calling for the results, since networking may be required to fulfill the request. Applications operate at 15 Hz, so this typically means waiting until the next 15 Hz execution. (In case of communications with far-away nodes, such as in another country, the application may have to be more patient than this.) When the call is made to collect the reply data for the entire request, the system returns the data right away if it is available. If it is not available, it attempts to be patient, since the newest data may have to come from other nodes. In the case that a request is made for only local idents, then networking is not used, and the reply data can be collected immediately, if desired. The key point here is that a Classic data request is made by a single call from an application. The reply data is available via a single call at some suitable time in the future. Settings are made for a single ident with one call, and networking is used to forward the setting message in the case that the ident references an external node.

As an illustration of how completely the system hides networking activity from a Classic protocol user, consider the parameter page application program that runs on the "little consoles" that are prevalent along the Linac gallery. This program was written before networking support of any kind was written for the Linac front ends. (This predates even the token ring network; the first Linac distributed system used SDLC protocol communications between Linac "local station" front ends, and a special node was used as a concentrator to connect via ethernet to a PDP-11 front end that was used to interface to the Acnet consoles.) When networking support was added to the underlying system, the parameter page program did not have to change. A user enters devices on the screen either by name or by node and channel number. The parameter page application merely retains a list of node and channel numbers for each of the 14 lines on the screen. It submits its request for periodic data from these devices by one request; later, it calls for the reply data at the requested rate. It doesn't have to be concerned with where the data came from, because the system takes care of it. When the call is made to collect the reply data, the system checks to see that all reply data from any external node is fresh; if it is not, it tries to wait for new data to arrive, up to a maximum time out of 50 ms past the start of the current 15 Hz cycle. For any external node reply data, the data is found from a buffer allocated to hold it when replies arrive from that node while that request is active. For local node reply data, the data is built at the time when the call is made to collect the reply data for the request, usually by sampling from the data pool that was refreshed early in the current cycle.