

Table Lookup Timing

IRM and PowerPC comparison

Tue, Jul 23, 2002

With the recent addition of table lookup logic to replace network table searches in both the IRM (MC68040-based) and PowerPC versions of the Linac front-end system code, we have an opportunity to compare the performance gained in both systems. This note describes the improvements in the execution time taken for these table search operations. Details on the new algorithms used for the table lookup support are described elsewhere. The motivation for doing this development was to reduce the impact on performance caused by slow access (about 1 μ s) to nonvolatile memory in the PowerPC systems. Implementation of the same algorithm for the IRMs was done as a pilot project.

Briefly, the table lookup scheme maintains relationships of node numbers with IP addresses, so that a native, Acnet, or pseudo node number can be used to look up an IP address, or an IP address can be used to look up any of the three node number types. Dynamic allocation of 64-address "subnet blocks" is based upon the IP addresses that are in use. (Some client nodes may have no node number relationships.) The key design objective was to make table lookup efficient. A secondary design goal was not to use too much memory. The present scheme is supported via a 64K byte data structure.

Timing diagnostics are included in the implementation so that the effectiveness of the scheme can be evaluated—and this document could be written. Besides the time needed to initialize the table lookup structure, the execution times of four routines were measured, each of which uses the low level table lookup functions to speed its execution. Thus the timing does not measure the table lookup functions themselves, but rather the total time spent in those routines that can take advantage of the new scheme. The four routines are called `PsNIPARP`, `IPNodeN`, `GtNodeN`, and `FindUDP`, each of which is described below in turn.

The first timing measurement is for initialization of the table lookup scheme, which populates the lookup table structure with the contents of the IPNAT table of native node number assignments to IP addresses, obtained from the DNS, and the contents of the Acnet TRUNK table of Acnet node number assignments to IP addresses, obtained from an Acnet server. Initialization in the IRM takes 14 ms; in the PowerPC, it takes 5 ms. This difference only occurs during system initialization, when the time used is insignificant, but it does illustrate one speed comparison between the two systems.

PsNIPARP

The `PsNIPARP` routine is called to construct a pseudo node number given an IP address and a UDP port number. This is done for all datagrams received by a node. The pseudo node number is a kind of 16-bit shorthand for a socket specification, in that it refers to an IPARP table entry that contains the IP address and the associated UDP port number. It is used internally for network message processing. The search time depends upon where the match is found in the IPARP table, so we use three different IPARP table indices to characterize the time spent to perform searches.

<i>Search</i>	#2	#42	#64	<i>Lookup</i>
IRM	16	41	54	23 μ s
PowerPC	7	70	103	3 μ s

Here we see the dependence on N for the search operations. The loop time dependence seems to be about 0.6 μ s for the IRM and 1.5 μ s for the PowerPC. It is only worse for the PowerPC because of its slow access time to nonvolatile memory. Note that the IRM performance using the lookup scheme suffers a bit if the match is found within the first 10 entries, but the dependence on N is removed. The PowerPC comes out well ahead, bettering the case of the search loop that finds a match right away.

IPNodeN

The `IPNodeN` routine searches the IPNAT for a match on a native node number to obtain an IP address, then calls `PsNIPARP` to construct the pseudo node number, so its time will always be

larger than `PSNIPARP`. It is needed only when sending a message to a native node number, not when sending replies to a request received from another node, where the pseudo node number suffices to identify the target node for the reply. Here, the indices refer to entries in the IPNAT.

<i>Search</i>	#2	#72	#160	<i>Lookup</i>
IRM	44	104	150	50 μ s
PowerPC	28	164	180	5 μ s

The search times show some apparent inconsistency, but that is due to the dependence on where matches are found in two different tables. These results only consider the position of the IPNAT entry. Since it calls `PSNIPARP`, though, it also depends on where a match is found in the IPARP table.

GtNodeN

The `GtNodeN` routine is called to search the IPNAT for a match on an IP address in order to obtain the native node number. This must happen for every reply message received by a node. This especially pertains to replies received by a server node, so it is very heavily used by the Linac data server `node0600`, which receives nearly all Acnet replies from the nodes contributing data in response to an Acnet request. The search time required depends upon the index of the matching IPNAT entry in each node.

<i>Search</i>	#2	#72	#160	<i>Lookup</i>
IRM	12	50	101	12 μ s
PowerPC	8	115	244	0 μ s

For timing the searches, three different indexes were used for the matching entries in the IPNAT. The dependence on where in the IPNAT the match is found made large differences in the time required for the search. For the IRM, the table lookup scheme effectively removed the dependence on where in the IPNAT the match was found. For the PowerPC, the search time nearly vanished. (Note that 0 μ s just means that the measured time was less than 1 μ s.)

FindUDP

The last search routine is `FindUDP`, which is called only when initiating an Acnet request. Its job is to search the TRUNK table for a match on an IP address, returning the Acnet node number. It would be needed by the Linac server `node0600` when forwarding a request for data from one other node. (If the request refers to more than one node, it is forwarded via multicast, so this call is unnecessary.) In this case, the index values refer to the TRUNK table entries, which can be lengthy. At this writing, there are only three trunks in use—trunks 9, 10, 11—but expansion to as many as 8 trunks is provided for the future. Each trunk has room for 256 entries, so the worst case index might one day reach 2000.

<i>Search</i>	#105	#416	#618	<i>Lookup</i>
IRM	41	106	150	13 μ s
PowerPC	164	632	935	0 μ s

The times listed here can be very long indeed, but the table lookup scheme all but eliminates the cost of this search. Again, the PowerPC time of 0 μ s only means less than 1 μ s.

Summary

One might ask whether the effort to implement the table lookup scheme is worth it. These front-ends operate in a 15 Hz environment. Their usual overhead is low, but they are also designed to operate under heavy load conditions, referring to the ability to support simultaneous data requests from many clients, all the while maintaining reliable 15 Hz operation. In a sense, table searches can be considered a waste of time, fraught with disappointment until a match is found. Several steps have been taken recently to eliminate as much table searching as possible in these front-end systems. The

table lookup scheme described here is a major part of that effort. Another change optimized the frequent searches for execution addresses needed when invoking dozens of local applications every 15 Hz cycle. A third example keeps an up-to-date “smart” copy of the nonvolatile CINFO (Channel Information) table in dynamic (fast) memory. Many years ago, a hashing scheme was implemented to support analog channel name lookup, thereby avoiding a search of the analog descriptor table. More recently, a network transmission diagnostic table was moved from nonvolatile memory into dynamic memory to improve performance. The system alarm scanning algorithm was also optimized for a minimum number of accesses to nonvolatile memory tables, only checking those entries that are currently enabled for alarm scanning.

For the PowerPC version of the system code, these efforts have served to largely mitigate the impact on performance of the slow nonvolatile memory. The PowerPC processor, using a 233 MHz clock, is much faster than the MC68040, which uses a 25 MHz clock. But when it stalls for 1 μ s for each access to nonvolatile memory, a lot of time can be wasted. In the future, additional schemes may be devised to further decrease performance dependence on this slow access time. For now, most nonvolatile table searches have been optimized in one way or another. Operating system kernel writers are known for devising algorithms that remove kernel performance dependence on N, so that system efficiency does not unduly suffer as a system grows to support more tasks, queues, etc. The efforts described here are done in that same spirit.