

UC Irvine

ICS Technical Reports

Title

Design considerations for the ARPA LNI name table

Permalink

<https://escholarship.org/uc/item/72s6r19h>

Author

Mockapetris, Paul V.

Publication Date

1978

Peer reviewed

7
614
C8
no. 12

Design Considerations for the
ARPA LNI Name Table

Paul V. Mockapetris

TR-92

UCI Technical Report 92

Revised May 1978

Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

This research was supported by ARPA, through the Information
Processing Technology Office, under contract
N00014-76-C-0954.



Abstract

Research in distributed computing at UC Irvine has included the development of ring communication hardware for the Distributed Computer System (DCS). [Mockapetris 78] The Ring Interface (RI), [Loomis 73] has recently been superseded by work on the Local Network Interface (LNI). [Mockapetris 77] The LNI includes implementation of ring communication systems as one of its goals.

The name table subsystem of both interfaces performs the functions necessary to recognize messages addressed to the attached host. These functions are divided into two parts. The first part is the comparison of the destination specification imbedded in the arriving message with the list of destinations stored in the name table. The second part of the NT subsystem allows the attached host to maintain the destination specifications stored in the name table.

The LNI name table design has been influenced heavily by the successes and shortcomings of the RI, and by the demands placed upon the RI by the DCS. We have corrected the deficiencies of the RI, and added new features. The bias has been toward an eventual implementation in LSI. Due to this bias, parts of the design are not optimized to TTL constraints and capabilities.

Name Table Issues

Name space facilities, such as the name table, are germane to any network. However, they are most useful in a system that multiplexes the use of the transmission media. In addition to ring systems, this class includes broadcast systems such as Ethernet [Metcalfe 76], loops such as SPIDER [Fraser 74], and various others.

Both the RI and the LNI are used to implement ring communication systems, although the LNI can be used in other modes. In a ring system, every host is coupled to the ring by one of the interfaces. The interface looks to the host like a fast full duplex device, consisting of two independent DMA channels (Input and Output) and associated status and command registers. To transmit a message, the host signals the interface to begin an output DMA transfer. When the transmission media becomes available, the message header and text are output through the DMA channel. The message makes a complete circuit of the ring, passing through each of the interfaces on the ring. The transmitting interface removes the message from the ring as it returns. Thus whenever a message transmission takes place, every interface "sees" the message. The input channel of the interface is logically "upstream" from the transmit side, and is capable of receiving the transmitted message as it returns, but before it is deleted. Those

interfaces that recognize the address in the message header will attempt to copy the message via the input DMA channel. Because the message is addressed to a process rather than to a host or interface, the interface must be provided with a list of processes in the attached machine. The name table subsystem manages this list.

It should be noted that the name table simply binds names to hosts; the use of process names is one interpretation.

The ring communication systems implemented by the RI and LNI also implement an implicit acknowledgement scheme that influences the name table subsystem design. When a message is transmitted, a receipt field is appended to the end of the message. (The receipt, CRC, and several other fields are invisible to the host and are seen only by the interfaces.) The receipt field consists of two bits, match and accept. These bits are set to zero when the message is transmitted. As the message passes through the interfaces on the ring, the message address is compared against the name table contents. If the address is found, the input side of the interface attempts to copy the message; the receipt field is left unchanged by interfaces that don't recognize the address. The copy operation can fail if the message CRC is bad, or if the input channel has not been enabled by the host, or due to several other causes. If the

copy is successful, a one is ORed into the accept bit. If the copy failed, a one is ORed into the match bit. If more than one interface recognizes the message address, the receipt field will be updated multiple times, although the ORing operations have no effect if the appropriate bit was already set. The transmitting interface records the returned setting of these bits in a status register that can be read by the host. This status information is used to drive sequencing protocols. This implies that the name table must always be ready to compare addresses, regardless of the host state. In addition, the name comparison cycle must proceed even if the host is attempting to concurrently update the name table contents.

In the RI, messages consist of the fields shown below:



The DPN field is 16 bits in length and specifies the destination process name of the message. The OPN is the process name of the process that sent the message. LEN is the length of the DATA segment. LEN is necessary because the messages are of variable length.

The RI attempts to copy all messages which have a DPN identical to any of the names in the RI name table. In the

RI, the name table is implemented as an array of 16 shift registers, each 16 bits long. Each of these slots is addressable by the host for the purpose of modifying its contents. As the message arrives (in bit serial form), each bit of the message DPN is compared in parallel with the head of each of the 16 shift registers. After each comparison, all of the shift registers are rotated by one bit. This comparison cycle is repeated 16 times. A flipflop associated with each of the shift registers is initialized to 1 before the match cycle starts. At each comparison, the flipflop is cleared if the bit of the message DPN does not match the corresponding bit in the shift register.

After all bits in the message DPN have passed, the 16 flipflops form a match vector of boolean values. An element of the match vector is 1 if, and only if, the corresponding name in the name table shift register exactly matched the message DPN. If any element of the match vector is 1, the RI attempts to copy the message.

Experience with the DCS system suggested several improvements to this scheme. These improvements enhance the capabilities of the overall system by allowing new features in the software.

It is difficult to specify the number of names required in a RI. While the present system does not often run out of

names in a given host, this is due to the fact that names are regarded as a very scarce resource. The original design of DCS [Farber 70], favored an implementation based on much smaller processes. The small number of names forced the use of larger process units.

Future roles envisioned for the LNI make any fixed size name table undesirable. Consider a host connected to the ring as well as to some other networks. This host is performing the functions of a gateway. A very convenient mode of operation would associate a name with each virtual data path on the other networks, or at least with each final destination. All of the names associated with a given gateway would route to the process managing the gateway. A large number of names would seem to be indicated. Another role envisioned for the LNI is to interface very dumb devices, such as disks or terminals, to the ring. This role would require only one or two names.

The obvious solution to this problem is to build the name table in an expandable fashion. The design approach is influenced by the desire to eventually implement the LNI in LSI. Clearly, one would like to have a keep the chip set necessary for a dumb host very small so as to limit the cost. The approach adopted is to include a small name table segment on the LNI chip, and to build a separate name table expander chip. As many of the expander chips as desired can

be daisy chained to the LNI chip. This decision is not obvious in the TTL implementation, but has been included in the design.

In order to make the name table expansion simple, the name table entries are no longer directly addressable. In the RI name table, commands from the host to load or delete a name are addressed to a specific name table entry. In the LNI name table, loads and deletes are handled by addressing name table entries associatively. The host specifies a name / mask pair to be either added or deleted. An add request directed at a full name table, or a delete request for a name that doesn't exist, generates an error status. When deleting names, the name / mask pair is compared exactly against the name / mask pair in the name table, instead of using the match rules for incoming messages.

Another set of enhanced features in the LNI are related to expanding the quality of the name space. In the DCS system, names have been used solely to specify processes. A software convention has been established in DCOS to manage the 16 bit process name as though it had three components. The 16 bit process name is broken down into a 4 bit process class, a 4 bit originating machine number and a 8 bit field used to generate unique names. For example, a process of class 6, created on machine 5 might have a sequence number of 33, thus its name would be 6533. While this scheme is

hardly sacred, it reflects the desire to give names some meaning. Such encoding is barely feasible in 16 bits. On the other hand, large fixed size names waste ring bandwidth. Variable size names would certainly complicate the hardware and software necessary. Some compromise is in order.

A second part of the name space problem is that of broadcast names. This facility is perhaps best explained by example. Consider a DCS process which provides access to a disk system. This process is an example of what the current system calls an I/O Handler (IOH). The IOH receives file access requests as messages from other processes. The IOH multiplexes the requests and sends back data records as messages. Furthermore, assume there are several of these IOHs in the system. A user process can communicate with any specific IOH via its unique process name. However, the user process may not know which IOH had a file of interest. One way to find the file is to ask each IOH on the system whether it has the file in question. This is a laborious process, and requires at least two messages per IOH. (First message: "Do you have file X?", Second message: "Yes I do. OR "Never heard of X.") Note also that this approach requires the user to know the names of all IOHs currently on the system.

The approach taken in the current version of DCS is to enter two names in the name table for each IOH. One name is

the unique name of the IOH. The other name is the broadcast name of the IOH. The broadcast names of all IOHs are the same, and are composed of a class component and a sequence component established by convention, and a machine number of zero. (By convention, there is no machine zero.) The user process can then send a message to the broadcast name, and the message will be received by all IOHs. This approach has the drawback that the system must consume one name table entry for each broadcast name. In the present system, broadcast names are only sparingly used because of this constraint. The general problem can be thought of as containing two aspects: fan-in and fan-out.

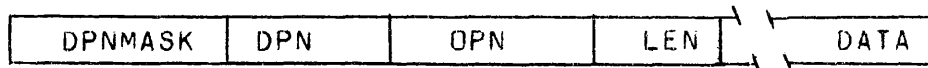
Fan-out is the broadcast message problem. We want a facility which enables the use of name specifications that only partially specify the destination process. In addition, this facility should not cause an excessive use of name table space. Suppose process names consist of four components: color, size, shape, and smell. A process might wish to send a message to those processes for which color=red and shape=round. This would be implemented by sending a message to color=red, size=*, shape=round, and smell=*. (Here "*" is used to represent "don't care".)

Alternatively, a given process might wish to represent itself as being a "one stop source" for all things blue, square and large. A similar problem is raised by a process

that has multiple sockets or ports that consist of a user id concatenated with a socket or port number. In this case we would like to create a name table entry that matches any of the sockets or ports for that user. This is the fan in problem.

LNI Message Addressing

The approach we have taken to this problem is to expand the name length to 32 bits, and to provide 32 bit "don't care" masks in both the message and each name table entry. The new message format looks like:



The DPNMASK, DPN, and OPN are all 32 bits in length. Each entry in the name table is composed of a 32 bit mask and a 32 bit name. The name table is implemented out of two 32 bit shift registers per entry. The rules for matching are:

1. For a message name specification to match the specification in any given name table entry, all 32 bits must satisfy the match rule.
2. For a given bit position:

EITHER

The corresponding bit in the DPNMASK or the corresponding bit in the name table mask must be 1

OR

The corresponding bits in the message DPN and the name table entry name must be equal.

Suppose that the high order 8 bits are the encoded process class component. A process can send a message to all processes of a given process class by setting the high 8

bits of the message DPN to be the encoding of the desired process class, and setting the low order 24 bits of the DPNMASK to all 1's. (The setting of the low order 24 bits of the DPN is irrelevant.)

If we want to create a process that has 8 sockets, we enter the high 29 bits of the process name in the name table name entry, and 29 bits of zeroes concatenated with 3 ones into the mask entry of the same name table slot.

Input Message Processing

The RI signals the arrival of a message by raising an interrupt request on the attached host. When the host services the interrupt, it senses the RI status to verify that the message is correct (CRC correct), and examines the message DPN to determine its process destination. Typically the operating system maintains a table of DPN / message queue address pairs. For every message arrival, this table must be searched to find the queue address where the message buffer is destined. An additional complication is that the message may be addressed to multiple processes in the same machine. Thus, the whole table must be searched. With the current maximum of 16 names this is not a significant problem, but could become one in future LNI systems. Not only is the name table arbitrarily long, but the test that must be done on each entry is more complex than a simple comparison. This problem could be especially severe for a gateway which typically would have a large name table space, if only a few eventual destinations.

The solution proposed for this problem is to associate an "ID" with every name in the name table. When a message is received and its status checked, the host would then receive from the RI the ID associated with each name table entry that matched the message. The basic implementation proposed is that the match vector be copied into a similarly

sized "mark" vector when a message is accepted. This is necessary because the match vector is volatile. When the host fields the RI input interrupt, it can ask the RI for the ID associated with the first mark bit set to 1. The LNI would then readout the corresponding ID, and reset that mark bit. Subsequent "read and reset" requests would receive different IDs, until all IDs were read out of the name table.

The semantics of the ID field would be up to the attached host. In the current system, the ID could be profitably be used to hold the message queue address corresponding to the name table entry. A gateway system might use the ID as the identification number of the destination network. A terminal cluster could use the ID for terminal number.

The ID could also serve the purpose of specifying name table entries to be deleted. If the ID is unique, any ambiguity as to the scope of a delete name table entry from the host would be eliminated.

However, this whole mechanism does add significantly to the amount of hardware necessary to realize the RI. For purposes of the first LNIs, the ID data paths and storage are not implemented. The hardware necessary to copy the match vector, and to read out the mark vector is included.

This enables the attached host to determine the number of name table entries that matched the message destination specification, but not to determine which entries.

Other Problems

There are other problems relating to the name table subsystem that we feel might be solved by future versions of the LNI. These problems are not addressed in the current LNI.

The message flow in the present system is uncontrolled. It would be desirable to be able to temporarily refuse messages addressed to a specific name. One scheme is to associate a flow control variable with each entry in the name table. If buffer space becomes critical in a given host, that host can turn off message reception for any set of names in the name table by setting flow control bits associated with each name. (Note that this refusal would be signalled to the sending process, through an enhanced receipt mechanism, as being due to flow control.) This facility, coupled with the appropriate system policies, would allow graceful processing of message transients.

Another possible improvement would be to include sequencing protocols in the R1. This would allow significant reduction in system programming requirements for small hosts. This effect is even more important in the interfacing of very dumb hosts (terminals) to a ring system. There are two problems with incorporating sequencing logic in the LNI. First, sequencing is done on the basis of

source / destination pairs. Thus an entry in the name table would be required for each source / destination pair, rather than one entry per destination. Secondly, a mechanism for initial connect protocol, or a non-reusable port structure similar to TCP [Cerf 78] would have to be implemented also. This would add considerable complexity. However, some such facility would be necessary if dumb hosts are to be directly connected to the ring, and require reliable transfer of messages.

A simple sequencing mechanism could greatly enhance throughput and reliability of broadcast message transmissions. When a broadcast message is sent in the present DCS system, the returned receipt can have four values, corresponding to the four possible combinations of the match and accept bits. If accept is set and match is zero, all the addressees of the message got it. If both of the bits are zero, no interface recognized the address in the message, and hence transmission is complete. If the match bit is set, and the accept bit is zero, a retransmission is performed. The problem arises when both bits are returned set. In this case one or more addressees copied the message, and one or more were unable to copy the message. What can be done? The transmitting host can retransmit the message. If it does so, it may eventually get a match of zero and accept of one back. This ends the

problem. (Any duplicates are discarded by the host's sequencing protocols.) However, if the broadcast message is addressed to a large number of processes, hence a large number of hosts, we can expect to see at least one of the interfaces unable to copy the message. This is simply due to the fact that the input side DMA channel has a dead time between the time that it finishes input of a message, and the time that the input interrupt routine of the attached host gets control, services the interrupt, and allocates a new buffer for input. Simple retransmission will likely result in more receipts with match and accept both set. These retransmissions also create similar problems for other hosts that are also trying to send broadcast messages.

The core of the problem is that no matter how many times we get both match and accept set, we don't know that everyone has received the message. It may be the case that the same interface is repeatedly refusing the message or it may be the case that everyone has received at least one good copy, and that retransmissions are serving no purpose, and wasting transmission bandwidth. If at all possible, we want to be sure that everyone copies the message because after at least one set accept bit has been seen, the sequencing protocols will fail if we don't successfully transmit to all. In addition, a guaranteed broadcast facility is very useful to processes.

One solution for this is the sequencing implemented in the LNI. Each name table entry has a sequence bit associated with it. Separate entries are used for each source / destination pair, and are created by message arrival if they don't already exist. In addition to the usual name comparison, a message is ignored if the sequence bit indicates a duplicate. (A sequence bit is also included in the message header.) When a message arrives, and is successfully copied, and its sequence bit is different from that in the message, the message is copied, and if the copy operation succeeds, the name table sequence bit is toggled. If the sequence bit is the same, the message is ignored, and the receipt is left unmodified. Thus each time the sender gets an accept bit set, it knows that the distribution of the broadcast message is at least becoming wider. The retransmissions that get ignored also do not cause interrupts and the concomitant dead time that is seen in the present system.

This approach could have considerable value in a system where broadcast messages are important. However, the increased size and complexity involved in the name table leads us to believe that this scheme will have to wait for a LSI implementation. It would be possible to implement this scheme in the present LNI without any modification by using one or more name bits as sequencing information. In this

case, a successful message reception would cause the host to delete the current name and add a new name to the table with an updated sequence field. The processes would have to insure that the name being used was only used by one sender. This is all somewhat awkward, and would only be practical if broadcast capability was very important.

Bibliography and References

- [Donnan 74] R.A. Donnan, and J. R. Kersey, Synchronous data link control: A perspective, IBM Systems Journal, vol.13, no.2, 1974, 140-162.
- [Farber 70] David J. Farber, "A Distributed Computer System", Technical Report 4, Sept 1970, Department of Information and Computer Science, University of California, Irvine
- [Farber 76] David J. Farber, and Kenneth Larson, Network security via dynamic process renaming, Proceedings of the 4th Data Communications Symposium, 1976.
- [Fraser 74] A. G. Fraser, SPIDER - An experimental data communications system, Proceedings International Conference on Data Communications, IEEE, June 1974
- [Cerf 78] Vinton G. Cerf, Jonathan B. Postel, Specification of Internetwork Transmission Control Program - TCP version 3, ISI Technical Report, Jan 1978
- [Loomis 73] Donald C. Loomis, Ring communication protocols, Tech Report 26, University of California, Irvine, 1973.
- [Metcalf 76] Robert M. Metcalfe and David R. Boggs, Ethernet: Distributed packet switching for local computer networks, Communications of the ACM, vol.19, no.7, July 1976, 395-404.
- [Mockapetris 77] Paul V. Mockapetris, Michael Lyle, David J. Farber, On the Design of Local Network Interfaces, Information Processing 77, Proceedings of the IFIP 77 Congress
- [Mockapetris 78] Paul V. Mockapetris, Experience with the Distributed Computer System (DCS), March 1978, Technical Report 116, Department of Information and Computer Science, University of California, Irvine CA.