AD-A060 214 CALIFORNIA UNIV IRVINE DEPT OF INFORMATION AND COMP--ETC F/G 9/2
A PROTOTYPE IMPLEMENTATION OF A PROTOCOL FOR NETWORK SECURITY.(U)
MAY 76 D J FARBER, K LARSON MDA903-75-G-0001
UNCLASSIFIED UCI-ICS-TR-84 NL

| OF |
AD
A060 214

END
DATE
FILMED
-12-78
DDC

UCI ICS UCI ICS

A Prototype Implementation of a Protocol for Network Security

by

David J. Farber and Ken Larson

Technical Report #84

University of California

Irvine

May 1, 1976

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under Grant MDA903-75-G-0001.

A Prototype Implementation of a Protocol for Network Security

The development of a protocol which achieves network security at the level of the communications system by dynamic process renaming has proceeded in two stages. First, the protocol was subjected to modelling and simulation which exposed misconceptions about the logical operation of the protocol. The results of this phase are summarized in [1,2]. This paper reports on the second phase of development in which a prototype system was created on the Distributed Computer System(DCS) at the University of California at Irvine. The prototype was created for the purpose of testing the protocol in a reasonably realistic environment. It is assumed throughout the paper that the reader is familiar with the protocol as presented by Farber and Larson in [1].

This report is divided into three sections. The first describes the DCS environment and its impact on design decisions made during the implementation of the prototype. The second describes the prototype implementation showing the close relationship between the protocol as modelled for analysis and the protocol prototype implementation. The final section summarizes the changes necessary to fully integrate the protocol prototype implementation into the DCS operating system

environment and suggests a design for a more flexible implementation of the protocol.

The DCS Environment

The DCS net is composed of three Lockheed SUE computers intercommunicating over a unidirectional data ring. The SUE computers are interfaced to the ring through a special communications device called a ring interface (RI) as shown in Figure 1. All interprocess communications are message based. Messages on the ring utilize process names rather than fixed hardware RI addresses as the message destination. This capability is achieved by including an associative store which functions as a process name table in each RI. When a message is received, the RI matches the destination address against the process name table to determine if the destination process is resident on the attached host(SUE). The DCS Software system is not machine dependent except for two processes:

1) the nucleus process which provides process scheduling and message routing facilities for the local resident processes

and

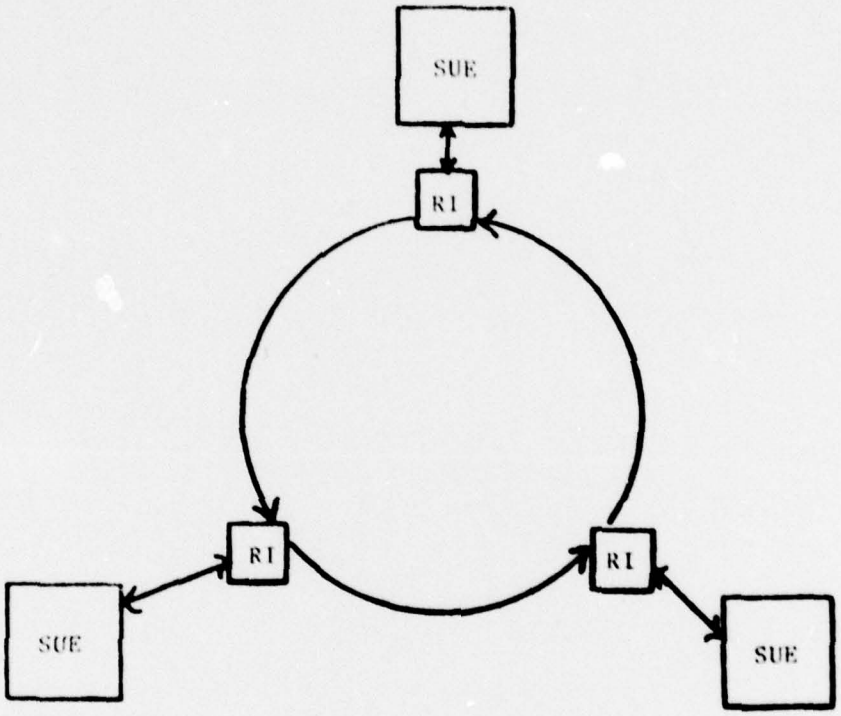

Figure 1. The DCS

2) the I/O handler process which provides an interface between the interprocess message system and the local I/O devices on that SUE.

The protocol under study is designed to supply security at the site to site level but not necessarily at the interprocess communications level within the host computers, although future investigations of the protocol may show that it is suitable for security at this level also. The current design is based on the existence of a separate communications processor which interfaces the host(SUE) to the communications system. The RI provides this service in the DCS environment, but the current RI cannot be programmed or easily modified. Thus, it is necessary to simulate the communications processor in software as a collection of site bound processes within each SUE. In the early stages, it was hoped that these processes might use the RI name table to eliminate unwanted messages by placing the receive names in the table. The name table, however, has only sixteen positions as many as six of which are consumed by system processes. There are too few remaining positions to support more than two send-receive links; thus, no effort is made to utilize the RI process name table.

The DCS operating system is not designed to interface to the simulated communications processor. A user process called LI can be used within the current prototype system to link to a "distant site". DCS, however, does not currently recognize the protocol control process (PCP) as a handler of the user teletype or terminal and will not transfer the name of the protocol control process to the created user process. As a result, a special user process has been created strictly for testing purposes. Changes to the DCS operating system which would allow total integration of the simulated communications processor are discussed in the final section.

The Prototype Protocol Implementation

The prototype implementation of the protocol centers on the simulated communications processor. The simulated communications processor is formed from two processes: 1) the protocol control process (PCP) which is an implementation of the protocol and 2) NETP which simulates hardware functions, performs allocation, and provides testing aids.

NETP is responsible for allocation of the channels through which processes can communicate with other hosts and the filtering and routing of messages. When NETP receives a request to open a channel to another host, it assigns a protocol control processes to the channel, informs the PCP of the user (requestor) process name, and sends the user process the name of the PCP assigned. It also records in the channel table the destination host and throughout the duration of the link maintains current entries of RN, ORN, and OORN as discussed in [1]. These entries are used to simulate the RI functions and refuse messages which are not addressed to a valid process name. The entries also provide a means of routing incoming net messages to the correct PCP process. All prototype communication over the ring is from a NETP at one host to a NETP at another host. NETP can be thought of as the "system" process necessary in a communications processor.

PCP is a direct implementation of the protocol as modelled [2]. PCP operates as a state machine with four states determined by two binary variables WTMSG and WTRSP. When both variables are zero (SEND) there are no messages expected from the net, i.e. no messages sent for which a response has not been received. In this state PCP waits for a message from the user process on the local host. If a reset message is received from the other PCP on the link, indicating

the other PCP is testing the link, an OK message is returned to maintain synchronization. When a message is received from the user, the PCP reformats the message and sends it to the local NETP which forwards the message to the NETP at the other host as dictated by the channel. The PCP sets the binary variable WTMSG and enters a state in which it waits for the response from the other host. If the PCP receives the returned message before a timeout occurs, it reformats the message and passes it to the user. The PCP then resets WTMSG and returns to the initial state to wait for a user response. If the PCP does not receive a response before a timeout occurs, the PCP sends a reset message to the other PCP to test the link and sets WTRSP to indicate that a response is expected from the PCP at the other host. When a normal message or response (reset-response or OK) message is received the associated variable is set to zero to indicate the expected response was received. If both are not set to zero before the end of the next timeout period, a major network failure is assumed and the PCP sends a "close" message to both the user process and NETP. NETP closes the channel by deleting the context associated with the channel.

The use of a separate PCP for each channel is chosen because the resulting code is more readable and easily understood than it would be if the process

multiplexed among the various channels. It also eliminates reproducing functions already present in the operating system. With future proposed modifications to the DCS software, it may be possible to use a single re-entrant instance of a PCP. In the final section, a similar implementation is proposed for future communications processor based designs.

System Operation

The current system operates as a vehicle for testing of the protocol. The prototype is comprised of four major processes. LI provides a means of linking a teletype to another host. ECHO is the only user process currently available at other hosts. It returns the user message after waiting for a predetermined length of time. NETP provides debugging and statistics gathering facilities. With these facilities one can print the messages sent and received by NETP and record the number of messages processed of each type: normal, reset, reset-response and OK. A feature also computes an "average response time" which is based on the average length of time between the transmission of a message by a channel and the receipt of a repsonse message by that channel. The protocol control process (PCP) is a direct implementation of the protocol as

described.

The test plan is simple. Observation of the message scripts generated by NETP during the operation of the prototype shows that the protocol operates as predicted by Farber and Larson[1]. The message addresses are generated by a pseudo-random number generator and exhibit a suitable degree of randomness. By varying the parameter in ECHO the message load rate can be changed. By varying a parameter in NETP an artificial ring error rate can be induced if the ring does not supply sufficient errors. NETP can be used to monitor the protocol response time under the various conditions of message load and ring error rates. This information provides a basis for determining timeout periods and an idea of expected performance under various conditions. The results follow the expected curve in which the rate of message flow on the network is correlated with the inverse of the delay in the user processes (ECHO and LINK). The relationship follows the dashed line in Figure 2. Since the bandwidth of the ring hardware is sufficient to handle any load created by the three hosts, there is no problem with interference from erroneous reset and response (reset-response and OK) messages. Thus, the waiting period for a timeout can be fixed at any reasonable length which is greater than the delay in the processes ECHO and LINK without significant effect on the message

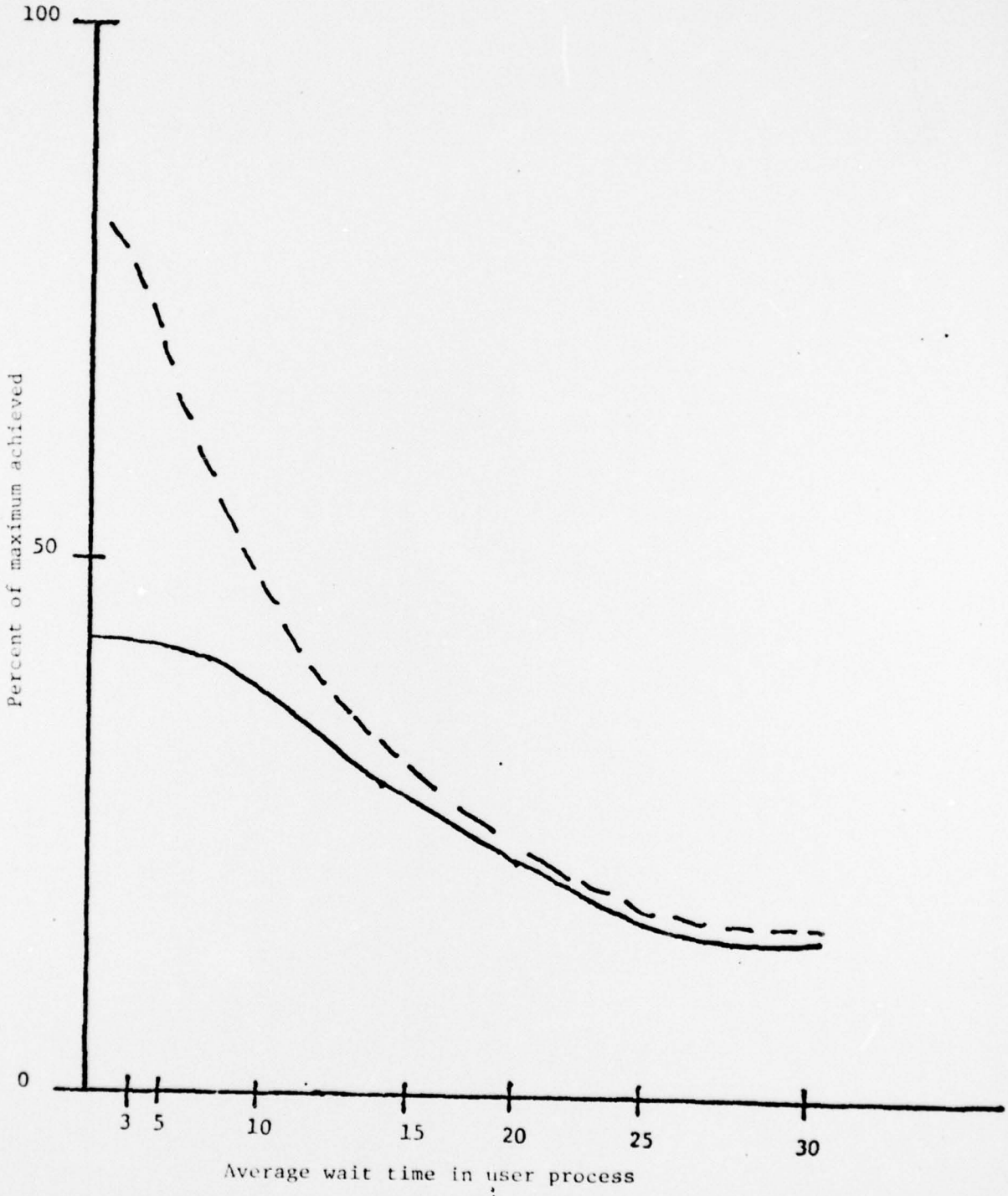
100
50
0
Percent of maximum achieved
3
5
10
15
20
25
30
Average wait time in user process

Figure 2

cate. This is not necessarily characteristic of all networks.

To gain a general picture of the expected performance of the protocol on other networks, the protocol is simulated as a Markov process in which time is divided into short intervals called epochs. The simulation of interprocess message traffic involves five pairs of communicating processes. Messages are queued for transmission on the network in a strict first-come-first-served manner on a network transmission queue at each host. The expected number of epochs that a message must wait for service from the net is defined by a parameter, QLOAD, i.e. the probability that a message is removed from the queue and transmitted to the destination process is given by 1/QLOAD. The transmission of the message from the network queue at one host to a process at another host is assumed to take a negligible amount of time and modelled as if it is instantaneous once the message is serviced from the network queue. A parameter LOAD determines the expected length of time that a message is processed in a user process before the process sends a message to the local network queue for transmission to the associated process at the other host. Thus, if a process has received a message, the probability that the process passes the return message to the network queue during an ensuing epoch is given by 1/LOAD.

The results of the simulation are shown in Figure 2. The verticle axis represents the message flow as a percentage of a "maximal" flow which would occur if a message were processed each epoch. The horizontal axis represents the LOAD parameter in epochs. The parameter QLOAD is fixed at 2, and the number of epochs before a timeout can occur and a reset message is generated is fixed at 32. The reset and response messages suffer from queuing delay but since they are processed in the communications processor, they suffer no delay at the host. In the graph the solid line represents the simulated message flow. The broken line represents the optimal message flow which occurs if reset messages are never generated and no messages are lost. The difference between the broken line and the solid line represents the loss of bandwidth due to network errors and the interference created by reset and response messages. The message flow is said to saturate at a specific value of LOAD, if as the LOAD increases beyond that value, the message rate does not significantly decrease. In the graph the protocol saturates with messages at about LOAD=25. This is also true when the simulation is run with a simulated rate of message loss of 5% and 10%. In all cases, the saturation occurs at nearly the same value of LOAD. While the simulated message loss does not effect the value of LOAD at which saturation is

reached, it does reduce the rate of message flow prior to saturation by a percentage roughly equal to the simulated rate of message loss, as would be expected. Message loss does not, however, appreciably effect the rate of message flow once saturation has been reached.

Future Plans

The total integration of the protocol prototype into the DCS operating system can be achieved in a "natural" way. In implementing the protocol control process, it was observed that PCP resembles the I/O handler process of DCS. With minor modifications the PCP can easily function as a pseudo-I/O handler as shown in Figure 3. The PCP need only recognize control commands sent to the I/O handler such as those which close the connection with the device or transfer the device to another process; the other messages can be passed to the actual I/O handler at the other host. The recognition of control commands could be implemented presently if the system process which loads user processes would recognize the protocol control processes as I/O handlers and pass the name of the PCP as the name of the user's terminal interface process. In the current environment user processes can be created at other hosts, but the user terminal will not be connected to the process. The DCS system is in the

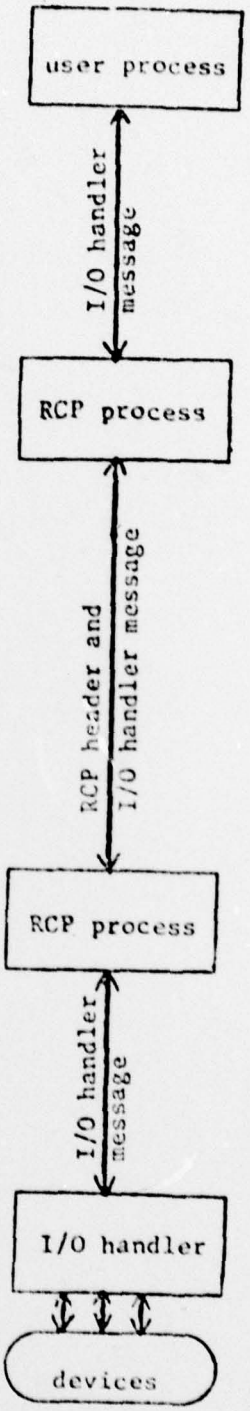
user process
I/O handler message
RCP process
RCP header and I/O handler message
RCP process
I/O handler message
I/O handler
devices

Figure 3

process of minor changes and until the shape of the new system is completely determined no attempt will be made to effect the change.

The prototype implementation in use is suited to DCS, but the protocol could be implemented in a more structured and efficient fashion on a dedicated communications processor. As described previously, the protocol is implemented as a state machine. The state of a channel can be determined by three binary variables: WTUSER, WTMSG, and WTRSP. WTUSER indicates that a message is expected from the user, and there are no messages expected from the net (WRSP=0 and WTMSG=0). WTMSG indicates that a message was sent to another host, but no message has been received in return, so a message is expected from the net. WTRSP indicates that a reset message was sent and a reset-response is expected but has not yet been received. The state changes are caused by receiving a specific message format. Due to the lack of a separate communications processor, the DCS prototype implementation required several environmentally dependent features. The RI name table is insufficient so routing and validation of destination names must be done in software. This requires the extra "destination" header, since all messages are addressed to the router process NETP. Since a premium is placed on the number of processes created, the router is also used to assign processes to

channels, although a separate protocol control process, PCP, is used for each channel. The router must also update the simulated name tables.

The DCS prototype functions adaquately in the DCS environment, but a much "cleaner" design of a prototype implementation is shown in Figure 4. With an expanded RI name table, the RI could validate names. The current NETP functions can be separated into four procedures: 1) the allocation and deallocation of channels, 2) the router which determines the "class" of the message and routes the message and the appropriate channel entry to the correct processing element, 3) the clock process which manages timeouts for the various channels and routes messages with the correct channel table entry to the appropriate processing element when a timeout occurs, and 4) an update process which manages the hardware and software(channel) name tables. For each transition defined in the protocol there is a processing element which takes a message and a channel table entry, performs the required processing associated with the transition, modifies the channel table entry accordingly including the state variables, and passes a message to either one of the previously described NETP processing elements or the user process. This design can increase efficiency by allowing more processing elements of a needed type to be added, without creating unnecessary elements which may not be

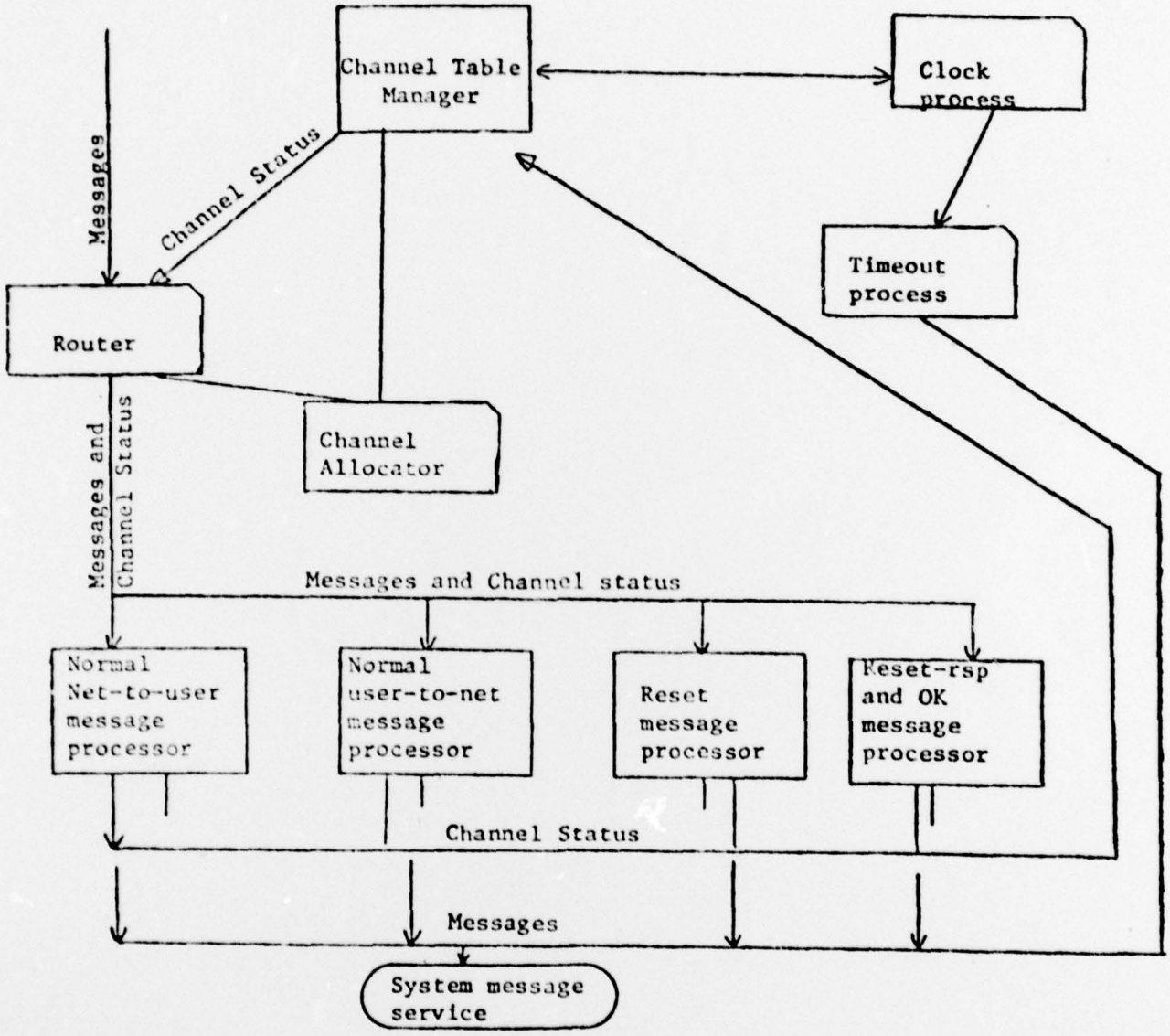
Channel Table Manager
Clock process
Messages
Channel Status
Timeout process
Router
Channel Allocator
Messages and Channel Status
Messages and Channel status
Normal Net-to-user message processor
Normal user-to-net message processor
Reset message processor
Reset-rsp and OK message processor
Channel Status
Messages
System message service

Figure 4

used, as happens when a complete process is assigned to each channel. It also provides a natural setting for multiprocessing. If a "bank" of microprocessors is available, each unit can be assigned the function of a processing element. If more processing elements of a specific type are necessary, more processors can be assigned that function. The router provides synchronization and may prove to be the limiting factor in such a system.

References

1. Farber,D.J. and K.C. Larson,"Network Security via Dynamic Process Renaming," Proceedings of the Fourth Data Communications Symposium, (October 1975).

2. Larson, K.C.,"Analysis of a Protocol Using a Token Flow Model," Technical Report No. 83, Department of Information and Computer Science, University of California, Irvine, (October 1975).