EIS-TR-77-6

A DESIGN OF PROCESS CONNECTION PROCEDURE ON COMPUTER NETWORK

by

Yoshihiko Ebihara
Shoichi Noguchi
Norman Abramson

March 15, 1977

INSTITUTE
OF
ELECTRONICS AND INFORMATION SCIENCE

UNIVERSITY OF TSUKUBA

EIS-TR-77-6

A Design of Process Connection Procedure on Computer Network

Yoshihiko Ebihara
University of Tsukuba

Shoichi Noguchi
Tohoku University

Norman Abramson
University of Hawaii

March 15, 1977

Abstract

It is important to express process connection procedures explicitly for researchers trying to connect computers, especially when a communication system is used across intercontinental distances. The objective of this paper is to show how a multilevel hierarchical structure of its procedures by means of finite state automaton graphs leads to easily understandable communication standards, complete specification. It also been implemented on HP2115 as one of ARPANET nodes.

1. INTRODUCTION

The growth of computer networks has recently become nation wide. International computer communication has been developed to permit persons at computer centers to access data and use interactive programs that exist and run on other remote computers. On the other hand, researchers trying to connect computers that are remote from each other across intercontinental distances faced,on occassion,various restrictions on implementation, communications and debugging of network control programs. Especially communication problems limit available debugging time between users and computer centers and the scheduled communication period of satellites often prevents each researchers from coming in close contact with. It also can be said that the process connection procedures of NCP (Network Control Program) which, provides facility for process-to-process communication over the computer network, are not exceptional. That fact is now recognized experimentally by the JALOHA computer network which is a Japan-Hawaii computer network using a Telex and a satellite link (ATS-1 and INTELSAT-4). From this point of view a simplified and resonable design for NCP is the prime design objective in a system to overcome the above restrictions. The basic objective of this paper is to apply state automaton and state graphs in the definition of process connection control and also to show how a multilevel hierarchical structure in connection procedures with state automaton graphs leads to easily understandable communication standards, complete and unambiguous specifications, embodying straightforward extensibility features. It has been implemented on the ALOHA-NCP which has become on ARPANET node.

2. DEFINITION OF CONNECTION PROCEDURES

In order to describe the process connection procedure, a few definition about a process, a subprocess, a link, a socket and a connection are needed as shown in the following:

Process: A process consists of subprocess (Pi_η) with hierarchical structure as shown in Fig.1, and is defined as a software module which executes a series of functions by communication with a subprocess of the remote host computer. Each subprocess with a pair of receive and send terminals becomes an executing subprocess when a pair of terminal

numbers is assigned to an operation of connection procedures. Further the communication between them is limited to processing within the same level of executing subprocess.

The necessity of a hierarchical structure in process connection can be prooved experimentally:

Some of the host computers differ from one another by type, speed, word length, operating system, and etc. Futhermore, a host may sometimes have a special character set. In order to support understandable cooperation of host-host communication in such an environment, many protocols have been specified, like that of the ARPA network. In particular, a NCP(Network Control Program) is provided for connection of independent processes in different hosts, control of the data flow, and several ancillary functions. One process deals with many kinds of data-types formats and functions during a connection period. This trend will increase in the future, because of rapid development of application protocols. However we can point out a corollary. Some specifications or mutual agreements have an effect on network communication and any host be able to talk with other remote host in the way based on the specifications, while some are concerned with the sphere of host-host level, and other are considered to be used only for subprocess-subprocess communication. Through interactive talk between Pi_1 and Pj_1as shown in Fig. 1, the details of definitions on functions, types or formats of data handled at the second level, Pi_2-Pj_2 connection, will be specified.

Accordingly, it is possible for subprocesses at level-2 to transfer and receive information data whose mode of expression may be newly specified without the restriction of limited standard character sets and formats for used at the level-1 communication. For example, subprocesses under the level-2 deal with voice, graphic and data base information. In the same way new definition and function for the third level of Pi_3 and Pj_3 subprocess communication will be, if necessary, formulated by interactive talds between Pi_2 and Pj_2 subprocesses. The original function of the process can be executed hierarchically with repetition of the above manipulation until the level-n connection.

These considerations lead to the introduction of a hierarchical structure in process connection which is built up by setablishing its own subprocess connections, step by step.

Socket number(S_n, R_n, S_n' or R_n', where n=1,2,3...):

A socket number is assigned by a host number and a socket number, and is attached to the terminal of the subprocess during execution of the subprocess connection. The same socket number in different hosts represents a different one so that it is uniquely identified over the network. However, a socket number assigned within a host has a narrow meaning here .

Link(l_n, where n=1,2,3...):

A link can be deffined as a logical path to couple two terminals of subprocesses in different hosts in simplex mode so that output from one terminal is input to the other, while the connection is fullduplex. A logical path must have a link number to establish a priority or distinguish it for a certain usage.

Subprocess connection:

A connection is a pair of links at the same level in different hosts. For example subprocess connection of level-1 means establishment of logical pathes between Pi_1 and Pj_1 subprocesses. Host 1 and 2 exchange relatively socket number (R_1) assigned dynamically by system of host 1 to the receiving terminal of Pi_1 and socket number (S_1') assigned by system of host 2 to the sending terminal of Pj_1. Simularly host 1 and 2 exchange socket number (S_1) and (R_1), relatively. These details of connection procedures are denoted later.

Process connection:

A process connection is defined as a sequential construction of subprocess connection from the level-1 to the necessary level-n. A process connection consists of the following three phases.

phase I : Initiation of process connection

phase II : Subprocess connection from level-1 to n

phase III : Subprocess disconnection from level-n to 1

On definitions mentioned above, a concrete process connection procedure is denoted in the next, while utilizing host-host control messages of ARPANET.

3. METHOD OF PROCESS CONNECTION

A process connection is initiated by a user initiator(UI). Both UI and a remote server initiator(SI) are connected to initialize start-up of process connection procedure, then the objective process initiator(PI) in the each host actually executes the required subprocess connection. We require images of host(i) and (J) in a network and should bring host(i)'s behavior into focus in the following explanation.

Function of UI:

User network commands let the system of host(i) know the selected process of host(i) and the remote process of host(j). The system chooses a unique free socket number(R_0), then hands R_0 to its UI via the CREATE(1) command. UI has the representative R_0 as its own receive terminal number(R_1') in turn which will be assigned to the first subprocess in the remote, control is handed to the selected PI.

Function of SI:

Each host is ready with common-socket number(D_0) to listen the request of process connection from other hosts. SI activates the selected process by observing the first control message(having D_0 in it) sent from host(j). In turn sending the representative receive socket number (R_1') to UI of host(j), the control is remove to the selected PI in host(i).

Function of PI:

PI exists inherently for each protocol in a host and establishes subprocess connection, from the level-1 to the level-n, of the selected process and disconnects each one.

3-1 CONNECTION PROCEDURES

We assume here that an access from host(i) to host(j) has occurred, assigning the process with the socket number D_0 for process connection. Each control message has been represented in three characters. Symbol (i) on the right shoulder of the symbol means message flow from host(i) to host(j) and (j) from host(j) to host(i). Each parameter within parentheses following the symbol should be referred to notes in table-1 and table-2.

phase-I (connection at level-0)

1. Host(i)'s UI sends a representative receiver socket R_0 by appointing the host(j)'s socket number D_0 and sends also the control

message ALL to allow data transmission on the link l_0. The format of control messages are expressed as; $RTS^{(i)}(R_0, D_0, l_0)$, $ALL^{(i)}(l_0, a, b)$.

2. Host(j)'s SI replies in turn with the control message STR to acknowledge; $STR^{(j)}(D_0, R_0, s_0)$.
3. SI chooses socket numbers S_1' and R_1' that will be assigned to the level-1 subprocess of selected process and sends $R_1'(=S_1'-1)$ as data information through l_0, by executing command SEND M(R_1').
4. Disconnection of both initiators means the end of process connection initialization. The close control message from UI's receive terminal is sent to SI's send terminal; $R\text{-}CLS^{(i)}(R_0, D_0)$.
5. Responding with host(i)'s control message, $R\text{-}CLS^{(j)}(D_0, R_0)$ is transferred as acknowledgement in turn.
   (Then SI and UI are released and are waiting for the next request.)

Phase II (connection at level-1)

6. Host(i)'s PI creates the executing subprocess Pi_1, at the level-1, to which assigning socket number R_1 and S_1 dynamically both socket number supposed to have relation with the representative R_0 like $R_1=R_0+2$, $S_1=R_1+1$
7. Transfer RTS to request the second level subprocess connection and ALL* to give space allocation to the remote subprocess; $RTS^{(i)}(R_1, S_1', l_2)$, $ALL^{(i)}(l_2, a, b)$.
8. Host(j)'s PI creates similarily the executing subprocess Pj_1 to which it will assign S_1' and R_1' socket number $S_1'=R_1'+1$.
9. A delivery STR in the form of $STR^{(j)}(S_1', R_1, s_2)$ is transmitted from host(j) to (i).
   (According to accomplishment of items 7 and 8, a notification of relative socket number R_1 and S_1' and establishment of the logical link l_2 are set up.)
10. $RTS^{(j)}(R_1', S_1, l_1)$ and $ALL^{(j)}(l_1, a, b)$ assigning for the link l_1 is transferred from host(j) to (i).
11. $STR^{(i)}(S_1, R_1', s_1)$ is in turn replied from host(i) to (j).
    (According to accomplishment of items 10 and 11, a notification of relative socket number S_1 and R_1' and establishment of the link l_1 are set up. Through procedure in items 7, 9, 10 and 11 the subprocess connection between Pi_1 and Pj_1 has been connected.)

Phase II (At this point various definitions and specifications necessary for the next level-2 subprocess communication are operated through the link l_1 and l_2, depending on behavior of application protocol. However we concerns only with exchange of mutual socket numbers used at the level-2 from the view of connection procedure.)

12. Host(j)'s socket number R_2' is transferred to the host(i) by command SEND M(R_2').
13. In turn host(i)'s R_2 is exchanged by command SEND M(R_2).

Phase II (subprocess connection of level-2)

14. Host(i)'s PI creates the executing subprocess Pi_2 assigning R_2 and S_2 calculated from the representative R_0, $R_2=R_0+4$, $S_2=R_2+5$.
15. Host(j)'s PI creates the executing subprocess Pj_2 assigning R_2' and S_2' calculated from the representive R_1', $S_2'=R_2'+1$, $R_2'=R_1'+2$.
16. Transmission of $RTS^{(i)}(R_2, S_2', l_4)$ and $ALL^{(i)}(l_4, a, b)$ activates establishing a logical link l_4.
17. $STR^{(i)}(S_2', R_2, s_3)$ is acknowledged from host(j) to (i).
(A notification of socket number R_2 and S_2', and establishment of l_4 have been done by executing items 16 and 17.)
18. $RTS^{(j)}(R_2', S_2, l_3)$ and $ALL^{(j)}(l_3, a, b)$ is sent to establish the logical link from host(j) to (i).
19. $STR^{(i)}(S_2, R_2', s_3)$ is acknowledged in turn.
(A notification of S_2 and R_2' has been executed, after establishing of l_3 and l_4, actual data is exchanged alternatively between subprocesses P_{i2} and P_{j2}.)

Phase III (disconnection of each level)

20. Close control message $R\text{-}CLS^{(i)}(R_2, S_2')$ is transferred from host (i)'s receive side to host(j)'s send side.
21. In the same manner the close control message $R\text{-}CLS^{(j)}(S_2', R_2)$ is sent from host(j)'s one to host(i)'s.
(According to completion of items 20 and 21, socket S_2' and R_2, and the logical link l_4 have been released.)
22. $S\text{-}CLS^{(i)}(S_2, R_2')$ is sent from host(i) to host(j).
23. $S\text{-}CLS^{(j)}(R_2', S_2)$ is sent in turn as acknowledgement of link termination.

(According to completion of items 22 and 23, socket S_2 and R_2', and the logical link l_3 have been released. Accomplishing items 20, 21, and 23, subprocess connection between P_{i_2} and P_{j_2} at the level-2 has been disconnected.)

24. R-CLS$^{(i)}$(R_1, S_1') is sent from host(i) to host(j).
25. In turn R-CLS$^{(j)}$(S_1', R_1) is sent from host(j) to host(i). (Socket number R_1 and S_1', and the logical link l_2 have been released through items 24 and 25.)
26. S-CLS$^{(i)}$(S_1, R_1') is sent from host(i) to host(j).
27. S-CLS$^{(j)}$(R_1', S_1) is sent from host(j) to host(i) in turn. (According to completion of items 24, 25, 26 and 27, the subprocess connection between P_{i_1}, and P_{j_1}, at the level-1 has been disconnected.)

In general similar subprocess connection procedure up to level-n may be operated repeatedly.

4. BLOCKING OF PROCESS CONNECTION PROCEDURE

The following process connection procedures consist of six foundamental blocks that have been expressed by the state automaton graphs. In graphs each state has been expressed by the circled marks ○ and ⬭, and more details of the explanation have been given in table 3. An arrow shows the next transition state. It is defined so that pural input events can be distinguished from each other in the case of a state with several branches to destination states. Symbols △ and ▭ represent the case of a branch destination reached from another block and identified by number within the △ or ▭ indicating the entry point numbered with same one in the other block.

4-1 U-INITIATE Block:

The function of this block is to take the main part of UI in the phase-I as shown in Fig. 2. The following shows description of each state in the U-INITIATE block gtaph.

INITIATE-1; This is the initial state in process connection procedure and the inner input event of CREATE command from the system causes creation of the executing subprocess P_{i_1} in the host(i).
T_1; The inner input event of CONNECT command from the system causes UI tranzent from state T_1 to T_2 and transmitt RTS$^{(i)}$ with R_0 and D_0 parameters and ALL$^{(i)}$ to the host(j).)T_2; Correct acknowledgment of

$RTS^{(i)}$ and $ALL^{(i)}$ from the host(j) causes UI set up bit(b) and message (a) count into the receive counter(RC). If these have not reached the host(j) correctly, received $[NA^{(1)}]$ brings UI to repeat transmitting these until $[AC^{(1)}]$ from the host(j) within a certain limit of time, while time-out causes UI to state F_1. T_3; Received $STR^{(j)}$ correponding the above $RTS^{(i)}$ and $ALL^{(i)}$ brings UI to state O_1, while time-out, with no reply for $RTS^{(i)}$ and $ALL^{(i)}$, brings UI into the forced disconnection. O_1; RECEIVE command from the system brings UI into state O_2 in which UI will find data of host(j)'s receive socket number(R_1') on data-message queue. O_2; Received data R_1' causes UI to store in its work area. Later PI will refer to it in operation at the level-1. O_3; Both $\underline{E(R)}$ and $\underline{\overline{E(R)}}$ events brings UI to state O_4 whether the receive counter (RC) is zero or not, as R_1 is the last data. O_4; CLOSE(1) command from the system UI send $R\text{-}CLS^{(i)}$ for a termination to the host(j). C_1; Received $[AC^{(1)}]$ for $R\text{-}CLS^{(i)}$ changes UI's status to C_2. If $[NA^{(1)}]$ returned from the host(j), UI repeats $R\text{-}CLS^{(i)}$ transmission within timo-out, until reception of $[AC^{(1)}]$ for $R\text{-}CLS^{(i)}$. C_2; UI goes to state C_5 with receiving $R\text{-}CLS^{(j)}$ as a reply for $R\text{-}CLS^{(i)}$. C_5; RELEASE command causes UI move to state R_1. R_1; The completion event of freeing resources used by UI processing, brings UI to come back to state INITIAL-2. The control will be handed to PI that works to realize the process connection, then UI will be released to accept the next request.

Detailed description of other block types has been omitted because of an analogous behavior in the praphs.

4-2 S-INITIATE Block:

Function of this block takes main part of phase-I procedure in SI side, and executes an initiating action for creating the executing subprocess under the control of SI that accepts process connection request from the source host (Reference to Fig. 2).

4-3 NORMAL OPEN-n Block ($n \geq 1$):

This executes subprocess communication between P_{j_n} and P_{i_n} at the level-n and deals with input/output data messages in half duplex mode. A data message from the host(i) to the host(j) is expressed by M(i) and conversely one from the host(j) to the host(i) by M(j). Format and code expression of data message and specification of functions transferred between P_{i_1} and P_{j_1} are followed in the way of these defined through the mutual communication at NOMAL OPEN-(n-1) block. An interrupt control

message and spetial control code imbedded in the data stream are used to indicate interruption of a subprocess. A special control code is illustrated by expression of M(jI) in data message unit.

At the level-n the host(i) is assumed to request an intrrupt from the host(j).

1. The interrupt request INT from the system lets host(i)'s PI transmitt the interrupt control message with the following format; $INS^{(i)}(1_m)$, where m=2n and n means the level-n subprocess connection (reference to notes of table-2). Then the interrupt data message M(iI) is sent by command SEND M(iI).

The host(i) is conversely assumed to accept the interrupt request from the host(j).

1. Receiving host(j)'s $INS^{(j)}$, RECEIVE command instructed from the system of the host(i) causes PI to inspect the data massage M(j) on the pending queue and incoming ones if it is a M(jI), then act on it, while throwing it away if it is not. After the accepting of M(jI), start signal INSG from the system means that the system had started to treat interrupt processing.
2. PI is ready to come back to state N_1 with occurrence of READY from the system after the interrupt management having been accomplished.

4-4 CLOSE-n Block($n \geq 1$):

Main operation of this block is to initiate closing, aborting and refusing the level-n subprocess connection between P_{i_n} and P_{j_n} and to acknowledge closing.

1. The command CLOSE(3) from the system causes PI to move from state N_1 to state C_1, while the close control message $S\text{-}CLS^{(j)}$ and $R\text{-}CLS^{(j)}$ brings PI from state N_1 to state C_3.
2. Upon state C_4 PI keeps waiting for all data message M(j) in the host(i) being treated completely by subprocess P_{i_n}, while on state C_2 PI assuming that there is no need to serve any operation for M(j).
3. All resources related with the subprocess connection at the level-n are freed by the RELEASE command, then PI returns to the open state N_1 of NORMAL OPEN-(n-1) block. However it is supposed to come back to state INITIAL-1, after resource-release at the level-1. (reference to Fig. 4)

4-5 CONNECT-n Block ($n \geq 1$):

This concerns with execution of subprocess connection of P_{i_n} and

P_{j_n} at the level-n. The left part in Fig. 4 shows initiation procedure of subprocess connection. Each symbol in the graph of CONNECT-n block includes parameters implictly and necessary receive socket numbers R_n and R_n' for the subprocess connection had been obtained through communication of the executing subprocesses $P_{i_{n-1}}$ and $P_{j_{n-1}}$. Send socket numbers S_n and S_n' are related to R_n and R_n', relatively, where $S_n=R_n+1$ and $S_n'=R_n'+1$. R_1 and R_1' are exceptions, being obtained from SI and UI operation.

4-6 FORCED CLOSE BLOCK:

Abnormal termination in the procedure brings PI to disconnect a subprocess connection forcely. After successful completion of resource release at the level-n, host(i)'s PI moves from state F_2 to open state N_1 in the NORMAL OPEN-(n-1) block. In the case of forced close at the level-1, it is assumed to return to state INITIAL-1.

5. PROCESS CONNECTION WITH HIERARCHICAL STRUCTURE

The block diagram has been illustrated to explain a general flow of process connection control procedure with hierarchical structure as shown in Fig. 5, where symbols of input/output events has been omitted in order to illustrate a control flow as a whole. However relationship of the input/output events is completely simular to that of explanation in each block. When time-out occurs in each block, it comes into the management of forced disconnection although time-out events are not put in the figure.

6. EXPERIMENT AND SOME RESULTS

In order to introduce the basic concept of process connection principle with automatic state diagrams, as described, into ARPANET, supporting program has been developed to cover incoming order of control messages from a host of ARPANET, and to guarantee that ARPAHET's host can communicate with as one of ARPANET, while it inerpreting even if it would be operating with other host based on the same procedure of itself.

On the basis of automatic process connection, the process control program has been implemented successfully on the ALOHA-NCP(Network control program for ALOHA system) installed on HP 2115, using the assember language. Though other functions required by ARPANET protocol also have been added to the new version, details has been omitted in this paper.

Through the experience with its implementation, it can be concluded that:

1. It is very easy and explicit to grasp the entire flow of process connection procedures, because of they being constructed from foundamental blocks, such as CONNECT-n, OPEN-n, CLOSE-n and FORCED CLOSE ones, which perform a subprocess connection at the level-n, and from the INITIATE blocks.
2. The connection procedures with hierarchical structure are easily adapted to expansion and modification in a subprocess-level layer because each layer can be mostly independet of specifications from each other and it may work reasonably for a connection procedures of newly developed application protocols.
3. As a subprocess at the level-(n-1) works under the control of the subprocess at the level-n in the same host. It is possible to obtain details of error or status information related with (n-1)-level subprocess that are on the desbugging, through communication between n-level subprocesses. The technique is very powerful as the means of developing and debugging new subprocesses in the environment of existing computer networks.
4. It is able to detect programing mistakes with the utmost rapidity and details by close investigation of the transition state diagrams of process connection control procedures. The author also believes such a technique of automatic process connection can be applied to implement microprograming for front-end mini computers.

7. CONCLUSIONS

A detailed description has been presented of a concept for implementation of process connection control in NCP that can operate in the realistic environment of computer networks.

Main principles can be described briefly that:

1. Mutual deffinitions of formats and data types to communicate and specification of functions operated are classified into network-level one, host-level one, subprocess-level one and so on, from the view points of how effect ranges of these deffinitions is covering over.
2. Under the consideration of the above, a process participated to communicate with different host in a Net can be divided into subprocesses and process connection is build up by establishing its own subconnection hierarchically.
3. Process connection procedures have been represented by automatic state-transition diagrams,
4. The connection control program on the implementation of ALOHA-NCP consists of foundamental blocks suitable for its implementation.

The technique of process connection procedures introduced is, we believe, one of fundamental ways in designing the NCP of computer networks.

* The ALL control message is sent from a receiving host(i) to a sending host(j) to increase the sending host(j)'s space counters. This may be sent any time the sending host's message counter a or bit counter b has been exhausted.

ACKNOWLEDGMENT

The authors would like to thank Dr.J. Oizumi of the University of Electro-Communications in Japan, R. Binder of UH and many staffs of the ALOHA project for useful discussions.

REFERENCES

[1] K. Fujita, T. Ikeda, S. Noguchi, R. Sato, J. oizumi, Y. Ebihara, F.F. Kuo and N. Abramson, A Japan-Hawaii Computer-Net---Telex and Satellite, Proc. the Seventh HICSS. on computer Nets, (Western Periodical Inc., UH, 1974).

[2] J. Oizumi, Y. Ebihara and S. Noguchi, The Pacific Area Computer Network, Information Processing Society of Japan, Vol. 16, no. 9 (1975).

[3] F.F. Kuo and R. Binder, Computer-communication by satellite: The ALOHA system, technical report. B 37-4, (The ALOHA system, UH, 1973).

[4] N. Abramson, Packet Swiching with Stellite, Proc. of NCC, (1974) 695-702.

[5] Y. Ebihara and M. Wilson, MENEHUNE ARPANET Driver, ARPA system Internal Documment, CCG/G-56(1974).

[6] L.Kleinrock, Performance Models and Measurements of ARPA Computer Network, proc. ONLINE conf., (1972) 61-85.

[7] L. Pouzin, presentation and Major Design Aspect of the CYCLADES Computer Network, Proc. of the Third Data Communication Symp., (Florida, 1973).

[8] R.E. Kahn, Resource-sharing Computer Communications Networks, proc. IEEE, Vol. 60, no. 11, (nov. 1972) 1397-1407.

[9] The File Transfer Protocol, ARPA Netword Information Center, no. 7813, (Nov. 1971).

[10] TELNET protocols, ARPA Network Information Center, no. 9348, (Apr. 1972).

[11] HOST/HOST Protocol for the ARPA Network, Bolt Beranek and Newman Inc., no. 8246, (Jan. 1972).

[12] Graphics Protocol, ARPA Network Information Center, no. 1538, (1973).

Table 1 INPUT SYMBOLS

CREATE(1)	:	Creation of User Initiator.
CREATE(2)	:	Creation of subprocess.
CONNECTION(1)	:	Request for UI initiation from system.
CONNECTION(2)	:	Connection request of subprocess from system.
CLOSE(1)	:	Termination of UI initiation.
CLOSE(2)	:	Termination of SI initiation.
CLOSE(3)	:	Desconnection of subprocess connection.
SEND	:	Transmission request for data message from system.
RECEIVE	:	Receive request of data message from system.
RELEASE	:	Resource release request used in subprocess connection from system.
LISTEN	:	SI's waiting for initiation request from UI.
READY	:	End of interrupt management.
INSG	:	Acceptance of interrupt management.
Δ	:	Status inquiry of subprocesses.
[NA]	:	Last message transmission error } [(1)] for control one.
[AC]	:	Successful last message transmission } [(2)] for data one.
*	:	Time-out for elapse of a certain time in state.
**(n)	:	Resource release of level-n.
E(S)	:	Event inkicating Send Counter (SC) being not zero.
$\overline{E(S)}$	:	Event indicating SC being zero.
E(R)	:	Event indicating Receive Counter (RC) being not zero.
$\overline{E(R)}$	:	Event indicating RC being zero.
*(D)	:	Complete processing of remained data messages.
AM(j)	:	Any messages from the host(j).
M(j), M(R_1')	:	Data message from the host(j).
N(jI)	:	Interrupt data message from the host(j).
INT	:	Interrupt request from the system.
$RTS^{(j)}$	:	Host(j)'s receiver-to-sender control message for a connection, $RTS^{(j)}(R_n', S_n, l_{2n-1})$, where R_n', S_n, l_{2n-1} are host(j)'s receiver and host(i)'s sender sockets, and link at the level-n, relatively.
$STR^{(j)}$	:	Host(j)'s sender-to- receiver control message for a connection, $STR^{(j)}(S_n', R_n, s_{2n})$, where S_n', R_n, s_{2n} are host(j)'s send and host(i)'s receive sockets, and a logical byte size, relatively.

$ALL^{(j)}$: Host(j)'s space allocation control message, $ALL^{(j)}(l_{2n-1}, a, b)$ where a and b are message and bit count assigned to l_{2n-1}.

$R\text{-}CLS^{(j)}$: Host(j)'s close control message to host(i)'s receive terminal of the level-n subprocess, $R\text{-}CLS^{(j)}(S_n', R_n)$.

$S\text{-}CLS^{(j)}$: Host(j)'s close control message to host(i)'s send terminal of the level-n.

$INS^{(j)}$: Host(j)'s interrupt control message, $INS^{(j)}(l_{2n-1})$.

NOTE

[ ] : Subnet control message.

$\underline{\quad}$: Underline of symbol means event from the system of host(i).

Nonexistence of parameters and control message is $INS^{(j)}(i_{-1})$, $RTS^{(j)}(R_0', S_0, l_{-1})$ and $S\text{-}CLS^{(j)}(R_0', S_0)$.

Table 2 OUTPUT SYMBOLS

$RTS^{(i)}$: Host(i)'s receiver-to-sender control message for a connection, $RTS^{(i)}(R_n, S_n', 1_{2n})$.

$STR^{(i)}$, $ALL^{(i)}$, $S\text{-}CLS^{(i)}$, $R\text{-}CLS^{(i)}$:
Explanations of the symbols are almost same as those of input symbols in table 2, replacing (i) by (j) and (j) by (i), except parameters, $STR^{(i)}(S_n, R_n', s_{2n-1})$, $ALL^{(i)}(1_{2n}, a, b)$, $S\text{-}CLS^{(i)}(S_n, R_n')$, $R\text{-}CLS^{(i)}(R_n, S_n')$

NO* : Subprocess troubles at a process connection request.

M(iI) : Host(i)'s data message.

DEQ : Buffer release of transmitted last message from queue.

ENQ : Acception of received message on queue.

D(RC) : Decrement received message and bits from Receive Counter.

D(SC) : Decrement transmitted message and bits from Send Counter.

S(RC) : Set RC a mount of message and bit space assigned by $ALL^{(i)}$.

S(SC) : Set SC a mount of message and bit space assigned by $ALL^{(j)}$.

NEG : Neglect all messages.

$INS^{(i)}$: Host(i)'s interrupt data message, $INS^{(i)}(1_{2n})$.

OK : Completion of creating UI or subprocess.

NOTE

Nonexistence of parameters and control messages is $STR^{(i)}(S_0, R_0', s_{-1})$ and $S\text{-}CLS^{(i)}(S_0, R_0')$.

parameters 1_m and s_m means:

1_m : m=2n, a link number assigned by host(i).
m=2n-1, a link number assigned by host(j).

s_m : m=2n, a logical byte size of data transferred on link 1_m.
m=2n-1, a logical byte size of data transferred on link 1_m.

where, m is interger number and n is interger number indicating level of connection.

Table-3 STATE SYMBOLS

INITIAL-1: Initial state for connection, where distinction between CONNECT(1) and LISTEN is assumed to be done.

INITIAL-2: Initialization end state.

LISTEN : Wait state for acception of connection request.

ACCEPT A_1: Wait for request of subprocess test inquiry from system.

A_2: Wait for reply [$AC^{(1)}$] of transmission $STR^{(i)}$.

OPEN

O_1: State able to receive data message.

O_2: Waiting for data message.

O_3: Check if there is enough space to continue receiving data.

O_4: Wait for CLOSE(1) command.

O_5: State able to send data message.

O_6: Wait for successful transmission $M(R_1)$.

O_7: Check if there is enough space to continue transmiting data.

O_8: Wait for CLOSE(2) command.

CLOSE

C_1: Wait for successful transmission of close control messages.

C_2: Wait for replies to close control messages from host(j).

C_3: Wait for resouce-release request.

C_4: Wait for end of received data message processing.

C_5: Wait for termination of resource release.

RELEASE

R_1: Wait for termination of resource release.

CONNECT

T_1: Wait for request of creating executing subprocess.

T_2: Wait for successful transmission.

NORMAL OPEN

N_1 : State able to receive and send data messages.

N_2 : Wait for data message M(j).

N_3 : Wait for data message M(I).

N_4 : Check if there is enough space to continue receiving.

N_5 : Check if there is enough space to continue sending.

N_6 : Wait for successful transmission $ALL^{(i)}$.

N_7 : Wait for $ALL^{(j)}$.

FORCED CLOSE

F_1 : Wait for request of resource release.

F_2 : Wait for termination of resource release.

INTEROUT

I_1 : Wait for successful transmission $INS^{(i)}$.

I_2 : Wait for RECEIVE command.

I_3 : Wait for M(JI).

I_4 : Check if there is enough space to continue receiving.

I_5 : Wait for successful transmission $ALL^{(i)}$.

I_6 : Check if there is enough space to continue sending.

I_7 : Wait for successful transmission $ALL^{(i)}$.

I_8 : Wait for starting signal of interrupt management INSG.

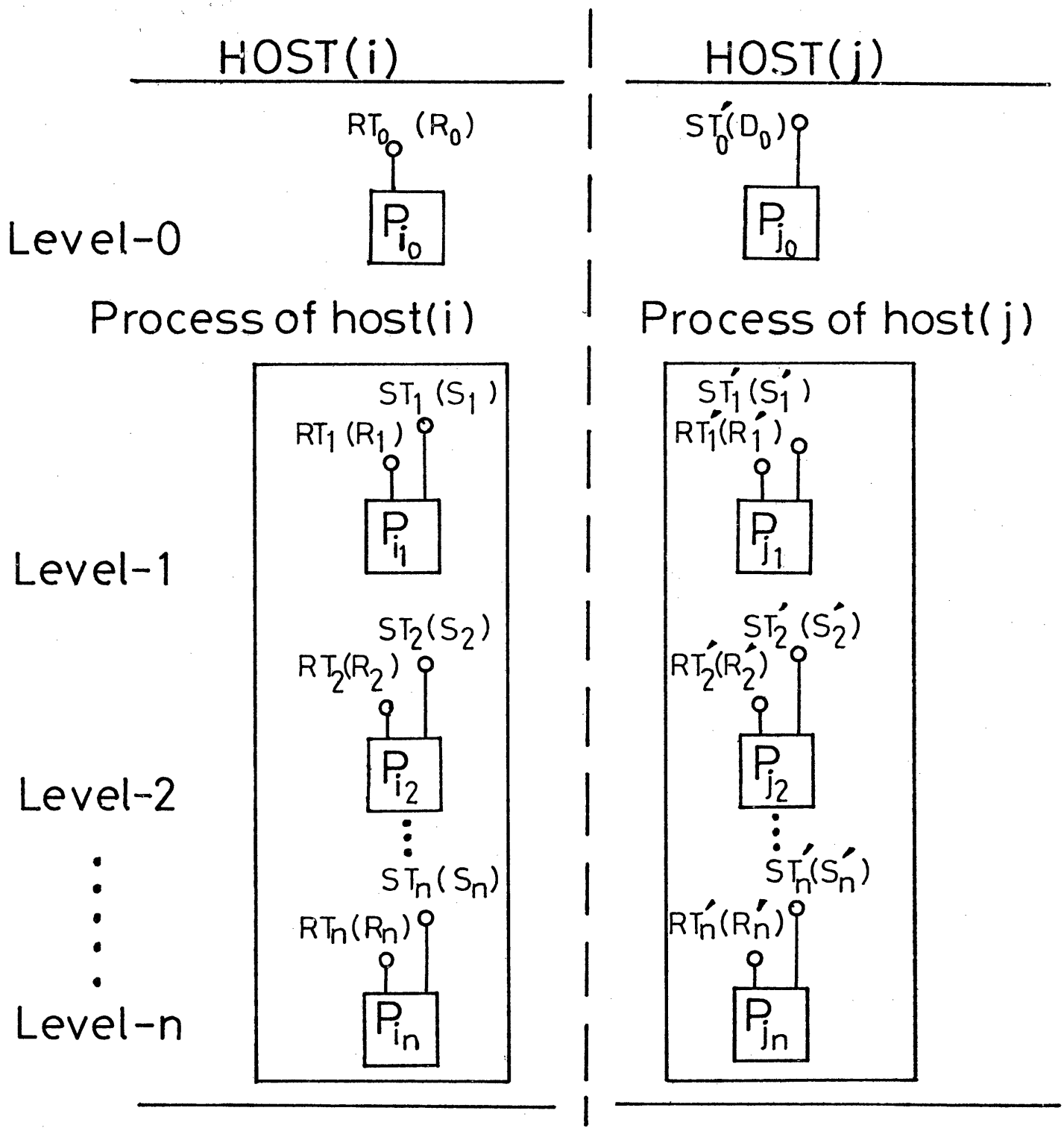
I_9 : Wait for completion of intrrupt management.

I_{10}: Wait for request SEND command.

I_{11}: Wait for successful transmission M(iI).

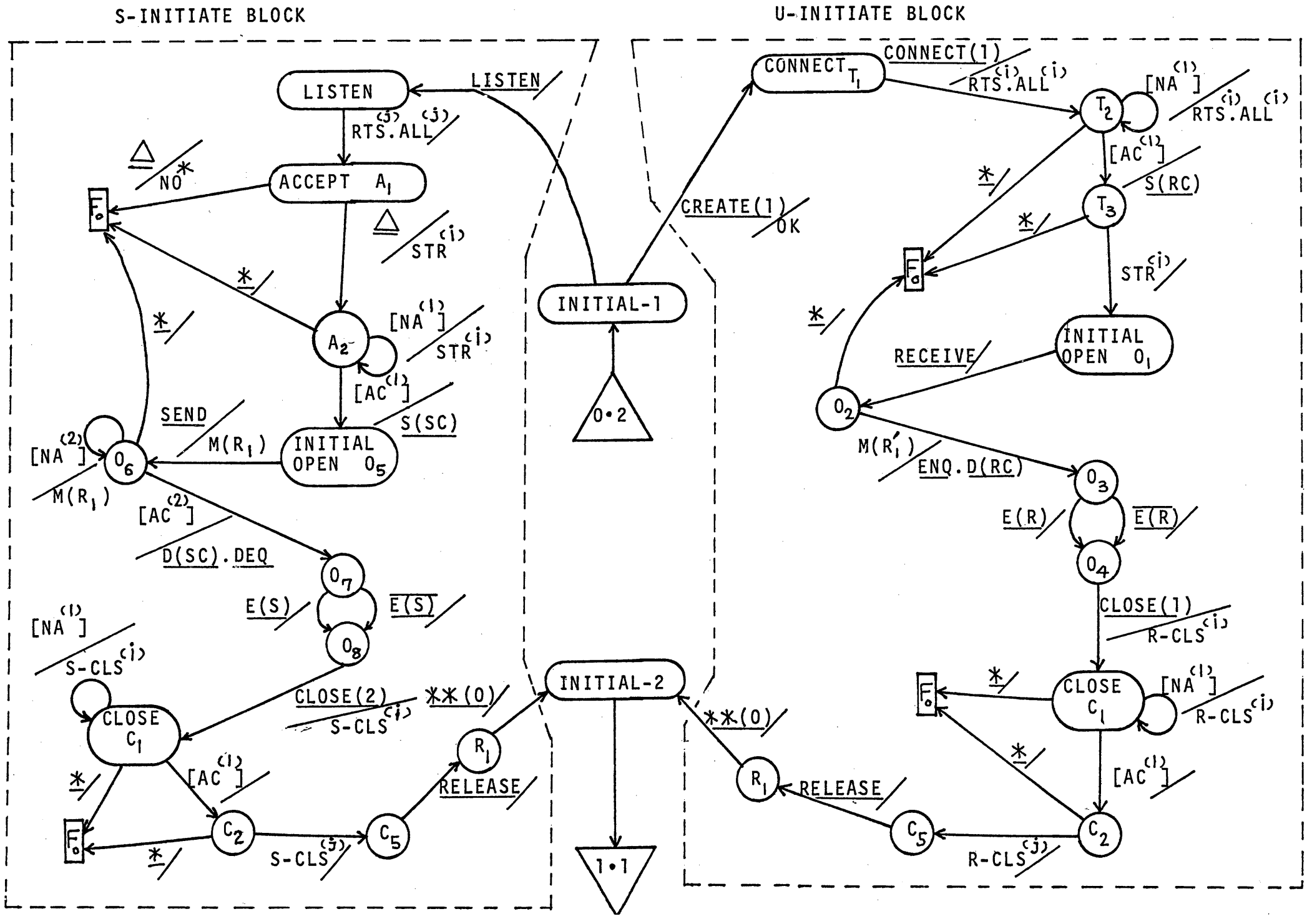
I_{12}: Check if thereis enough space to continue sending.

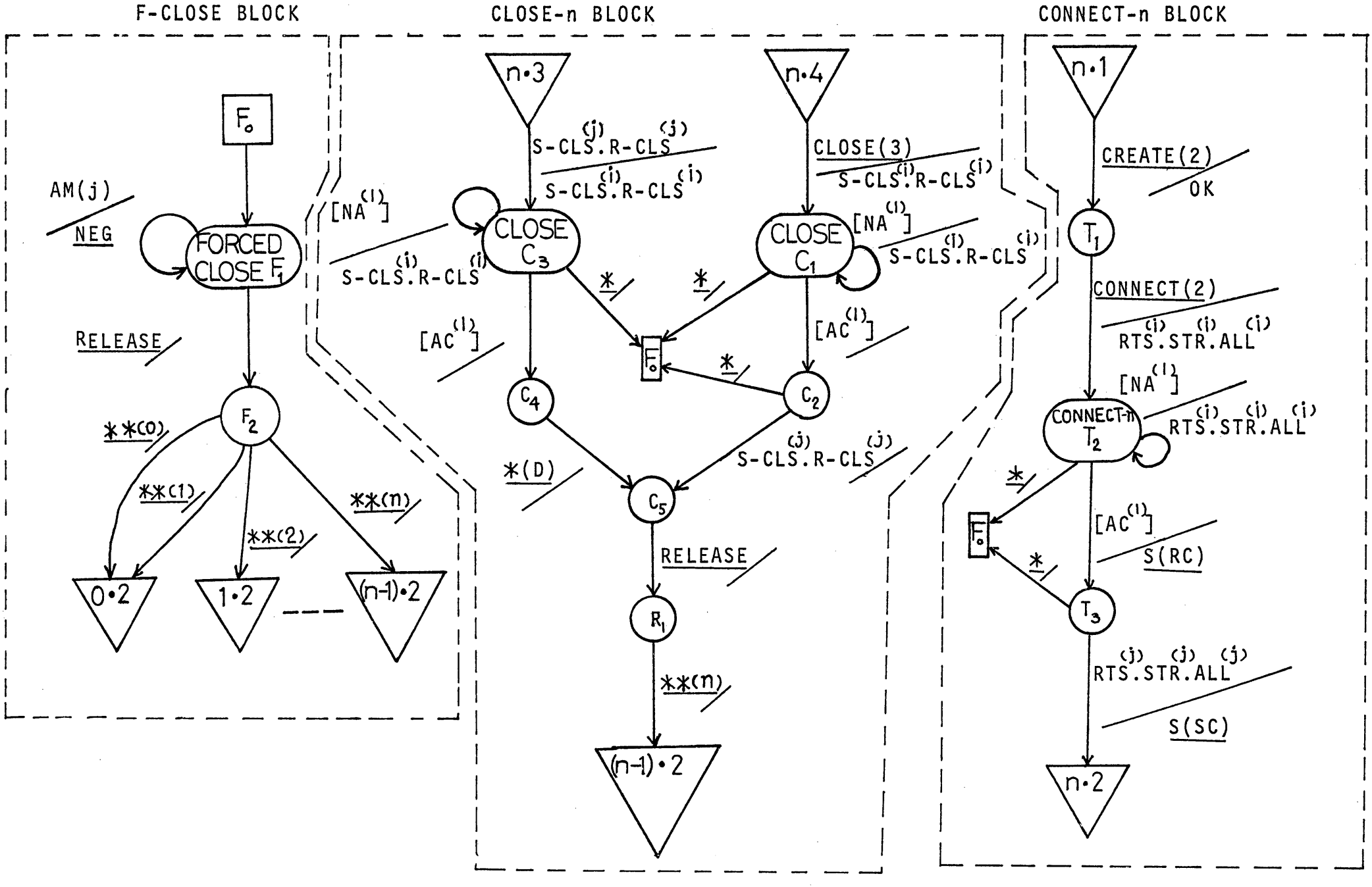
I_{13}: Wait for space-a allocation control message $ALL^{(j)}$.



For n=1,2,3,....
RT_n :Receive terminal at level-n, ST_n :Send terminal at level-n
R_n :Receive socket number assigned to RT_n at level-n
S_n :Send socket number assigned to ST_n at level-n
P_{in} :Subprocess of host(i) at level-n
P_{jn} :Subprocess of host(j) at level-n
(Apostrophized RT_n, ST_n, R_n and S_n express those of host(j)

For n=0
RT_0 :Receive terminal of host(i) at level-0
ST_0 :Send terminal of host(j) at level-0
R_0 :Receive socket number of host(i) at level-0
S_0 :Send socket number of host(j) at level-0, $S_0 = D_0$
P_{i_0} :User initiator
P_{j_0} :Server initiator


S-INITIATE BLOCK
U-INITIATE BLOCK
LISTEN
LISTEN
RTS.ALL(j)
ACCEPT A1
NO*
STR(i)
A2
[NA(1)]
[AC(1)]
INITIAL OPEN O5
SEND
M(R1)
O6
[NA(2)]
[AC(2)]
D(SC).DEQ
O7
O8
E(S)
E(S)
S(SC)
CLOSE(2)
**(0)
S-CLS(i)
CLOSE C1
S-CLS(j)
C2
C5
R1
RELEASE
INITIAL-1
0•2
INITIAL-2
1•1
CONNECT T1
CONNECT(1)
RTS.ALL(i)
T2
T3
[AC(2)]
S(RC)
CREATE(1)
OK
INITIAL OPEN O1
RECEIVE
O2
M(R'1)
ENQ.D(RC)
O3
O4
E(R)
E(R)
CLOSE(1)
R-CLS(i)
R-CLS(j)


F-CLOSE BLOCK
CLOSE-n BLOCK
CONNECT-n BLOCK
F0
AM(j)
NEG
FORCED CLOSE F1
RELEASE
F2
**(0)
**(1)
**(2)
**(n)
0•2
1•2
(n-1)•2
[NA(1)]
S-CLS(i).R-CLS(i)
n•3
S-CLS(j).R-CLS(j)
CLOSE C3
[AC(1)]
C4
*
F0
*
C2
*(D)
C5
S-CLS(j).R-CLS(j)
RELEASE
R1
**(n)
(n-1)•2
n•4
CLOSE(3)
S-CLS(i).R-CLS(i)
CLOSE C1
[NA(1)]
S-CLS(i).R-CLS(i)
[AC(1)]
n•1
CREATE(2)
OK
T1
CONNECT(2)
RTS(i).STR(i).ALL(i)
[NA(1)]
CONNECT-n T2
RTS(i).STR(i).ALL(i)
*
F0
*
[AC(1)]
S(RC)
T3
RTS(j).STR(j).ALL(j)
S(SC)
n•2

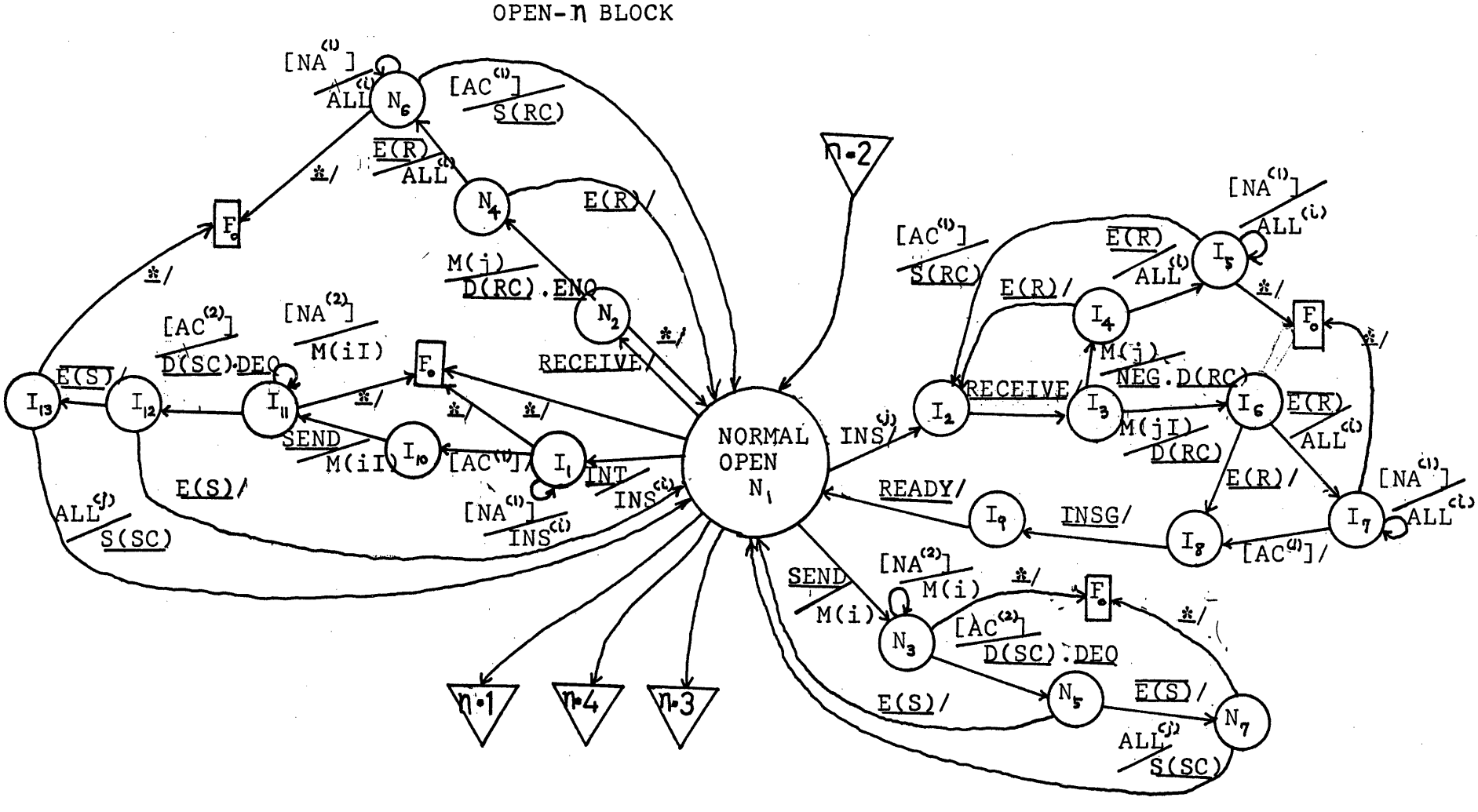
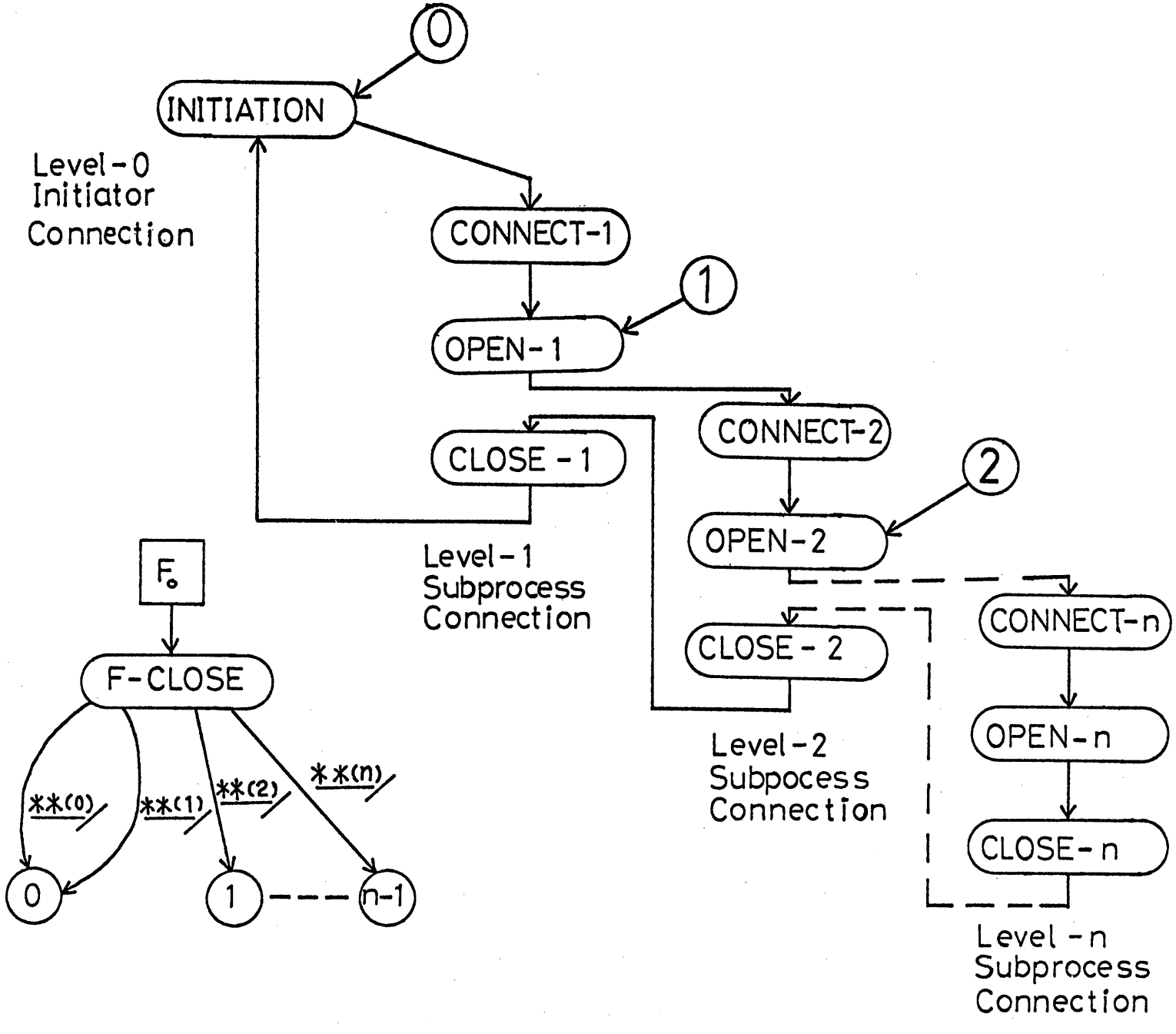


Fig.3 Open-n Block


0
INITIATION
Level-0
Initiator
Connection
CONNECT-1
1
OPEN-1
CLOSE-1
CONNECT-2
2
OPEN-2
Level-1
Subprocess
Connection
F₀
F-CLOSE
**(0)
**(1)
**(2)
**(n)
0
1
n-1
CLOSE-2
CONNECT-n
OPEN-n
CLOSE-n
Level-2
Subpocess
Connection
Level-n
Subprocess
Connection

INSTITUTE OF ELECTRONICS AND INFORMATION SCIENCE
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER EIS-TR-77- 6

TITLE

A Design of Process Connection Procedure on Computer Network

AUTHOR(s)

Y.Ebihara

(Institute of Electronics and Information Science,Univ. of Tsukuba)

S.Noguchi

(Research Institute of Electrical Communication,Tohoku Univ.)

N.Abramson

(The ALOHA System,Department of EE.,University of Hawaii)

REPORT DATE	NUMBER OF PAGES
March,15,1977	25
MAIN CATEGORY	CR CATEGORIES
Communications	3.81

KEY WORDS

Connection Procedure,Process,Subprocess,State graphs,
Link,Socket.

ABSTRACT

It is important to express process connection procedures explictly for researchers trying to connect computers,especially when a communication system is used across intercontinental distances. The objective of this paper is to show how a multi-level hierarchical structure of its procedures by means of finite state automaton graphs leads to easily understandable communication standards,complete specification.
It have also been implemented on HP 2115 as one of ARPANET's nodes.

SUPPLEMENTARY NOTES