Graph Modeling of Computer Communications Protocols

Jonathan B. Postel

and

David J. Farber

Technical Report #85

May 1976

Jonathan B. Postel
Computer Science Department*
University of California
Los Angeles, California

David J. Farber
Department of Information and Computer Science
University of California
Irvine, California

Current Address:

Augmentation Research Center
Stanford Research Institute
Menlo Park, California 94025

Introduction

The design of computer-to-computer communications protocols for networks of computers is one of the challenging problems facing the computer scientist. In the last few years, a number of large scale computer networks have been discussed and several of these are in various phases of implementation. It is clear that the design of the communications protocols for these networks is a difficult problem (though the evidence for this is mainly in internal memoranda, see [CARR70], [CROC72], [FARB73], [FRAN72], and [HEAR70]). Many of the problems in designing these communications protocols, which involve asynchronous parallel processes, are due to the difficulty in recognizing all of the possible orderings in which the significant events can occur and the possible execution sequences which arise from these orderings.

This paper draws from experience with the development of a computer network sponsored by the Advanced Research Projects Agency (ARPA) called the ARPANET [ROBE70], and the development of the UCLA Graph Model of Computation [ESTR63], especially the advances made by Gostelow [GOST71] and Cerf [CERF72c] which gave this model of parallel processing additional analytical power. One especially attractive aspect of this modeling technique is the existence of a computer program for testing for the property of proper termination. Proper termination indicates that the process modeled by the graph is in a certain sense well behaved; for example it has no deadlocks. These developments suggest that communications protocols can be modeled by the UCLA Graph Model in a useful and revealing way.

SECTION 1 -- A Communications Protocol

A communications protocol must deal with several sometimes contradictory issues. Among these are error control, flow control, multiplexing, and synchronization. The approaches taken to the issues have implications affecting retransmission frequency, buffering requirements, throughput, and delay. In contemplating all of these issues, two "zero level" issues must be kept in mind: deadlocks, and critical race conditions.

Deadlocks are situations in which two or more parties (processes, hosts) are waiting for a resource (command, signal) before proceeding (with their tasks) and that resource can only become available by one or more of those same parties proceeding. Critical race conditions are those situations where the outcome of a sequence of

(supposedly independent) operations is dependent upon the order in which the operations occur.

In the following section, the error control aspect of an example protocol is developed, then in Section 3 the example protocol is modeled.

A basic requirement at any level in a communications system is error control. Generally in the type of networks being considered the communications subsystems are designed to be very reliable. However, there may be interface hardware and some short transmission paths that are unchecked by the communications subsystem, also there may be temporary outages in the communications subsystem or messages lost due to congestion (and even errors which occur with very low probability can be costly); thus error checking is still required. A simple scheme for error control is to use a checksum attached to the message which the sender calculates and attaches to the message and the receiver calculates and compares with the checksum transmitted with the message. If there is a discrepancy, then an error has occurred. It is possible to use error correcting codes, but their discussion is beyond the scope of this presentation.

The sender must have some way of knowing that the message arrived at its destination correctly or incorrectly. A simple way to provide this knowledge is to have the receiver acknowledge each correctly received message. The sender can then assume that if no acknowledgement is received by the end of a timeout period then either the message was afflicted with errors or was undelivered. In this case the sender retransmits the message. Note that the acknowledgement must also be checksummed since it is transmitted via the same media as the message, and therefore subject to errors in the same way.

The protocol proposed thus far is not sufficient. It fails when the finite time for message delivery and the variability of that time is considered. If the delivery time in one instance is approximately equal to the timeout period, then the sender can decide to retransmit just as the receiver decides to send the acknowledgement (of the first copy). These then pass each other in the communications network. The result is that the receiver accepts two copies of the same message and the sender gets an extra acknowledgement. Further one or the other of the acknowledgements could be afflicted with errors and then the sender would not detect the problem either. There must be added to the protocol some means of identifying a message; for convenience, numbers will be used. Thus, if the sender attaches a sequence number to each message the receiver can

check to be sure it has not previously accepted a copy of that message. Because there must be some maximum sequence number, M, the sequence numbers will be used cyclicly, that is after sequence number M is used the sequence is restarted at zero. Another way to view this is to say that messages are counted indefinitely and that the sequence number is the message count modulo M+1. This protocol is illustrated in Figure 1.

In the program-like description below, if an "if clause" is false then fall through to the next statement (usually an if) at that same level (of indentation). If a "go to" directs the reexecution of a statement such as "if Ack received" then it is intended that a new instance of an Ack be received before the statement can be reexecuted.

The drawing is a state machine. In the notation accompanying an arc the statement above the bar indicates the input required to activate the arc, and the statement below the bar indicates the action taken while transiting the arc. A lambda indicates either no input or no action.

The sender starts in "initialization" state A, and moves to "send message" state B by setting N to zero (step S0). From state B the sender moves to "wait for Ack" state C by sending message N and starting a timer (S1, S2). In state C the arrival of an acknowledgement with a bad checksum (S3, S4, S5) results in continuing in state C. In state C the arrival of an acknowledgement with a good checksum (S3, S6) results in moving to "decision" state D. If the timer expires (S12, S13) the sender moves from state C to state B. If the sequence number of the received good acknowledgement is the same as the sequence number of the message sent then the sequence number is incremented and the sender moves from state D to state B (S7, S8, S9). If the received sequence number is not the same as the sent sequence number the sender moves from state D to state C (S10, S11).

The receiver begins in "initialization" state E, and moves to "wait for message" state F by setting M to zero (step R0). If the messages is received with a bad checksum the receiver remains in state F (R1, R2, R3). If the message is received with a good checksum the receiver moves to "send Ack" state G (R1, R4). The receiver moves from state G to "decision" state H by sending an acknowledgement (R5). If the sequence number received is not the expected one (R6, R7) the receiver moves to state F. If the sequence number is the expected one the receiver increments the sequence number and moves to state F (R8, R9, R10).

Sender	Receiver
S0: N=0	R0: M=0
S1: Send Msg(N)	R1: Rcv Msg(J)
S2: Start Timer	R2: if checksum = bad
S3: if Ack(I) received	R3: then go to R1
S4: if checksum = bad	R4: if checksum = good
S5: then go to S3	R5: Send Ack(J)
S6: if checksum = good	R6: if J not equal M
S7: if I = N	R7: then go to R1
S8: then N=N+1	R8: if J = M
S9: and go to S1	R9: then M=M+1
S10: if I not equal N	R10: and go to R1
S11: then go to S3	
S12: if Timer runout	
S13: then go to S1	

A Communications Protocol

A parameter to be considered in this mechanism is the number of different unacknowledged messages allowed. It has been shown that if this number is one, the sequence number need be only one bit [BART69]. However, when only one message is allowed to be outstanding the bandwidth of the communications network may be inefficiently used [METC73].

This example protocol while addressing only the issue of error control is sufficiently developed to lead to an interesting graph model and analysis.

SECTION 2 -- The UCLA Graph Model

The UCLA Graph Model has developed from the acyclic bilogic directed graph model introduced by Estrin [ESTR63]. The current model consists of arcs and vertices where generally speaking the vertices represent computations and the arcs represent the flow of control between computations. At each vertex there is either "exclusive or" (EOR) or "and" (AND) logic between the input arcs, and similarly EOR or AND logic between the output arcs. EOR input logic means that if any one of the input arcs is enabled (carries a token) the vertex can execute. AND input logic requires all input arcs to be enabled before the vertex can execute. EOR output logic indicates that when the vertex completes execution it will enable (place a token on) one of the output arcs. AND output logic indicates that when the vertex completes execution it will enable all the output arcs. A vertex executes by removing token(s) from its enabling input arc(s) and placing token(s) on its output

arc(s) as indicated by the output logic. If more than one of a set of EOR input arcs is enabled the vertex chooses randomly between the enabled arcs as to which token to remove. On EOR output logic, the vertex may decide which arc to enable based on an interpretation associated with the graph; in an uninterpreted graph the choice is random. The UCLA Graph Model also allows complex arcs, i.e. arcs with multiple tails or heads.

A UCLA Graph Model representation of a simple exchange of a message and reply is shown in Figure 2, where the vertices have the meanings:

1 = begin exchange
2 = send message
3 = receive message
4 = send reply
5 = receive reply
6 = end exchange

and the arcs have the meanings:

S = ready for exchange
A = ready to send message
B = ready to receive message
M = message available
C = read to receive reply
D = read to send reply
R = reply available
E = reply received
F = reply sent
X = exchange completed

A number of useful constructs and concepts have been associated with the UCLA Graph Model, the current development of which is called the Graph Model of Computation (GMC). Tokens are used in the UCLA Graph Model to analyze the flow of control through the graph as the vertices execute. A mechanism for testing GMC's for certain properties is called the Token Machine (TM). One of these constructs is the Computation Flow Graph (CFG) which is a state transition diagram where the states are named corresponding to the token placement in the GMC. The CFG for the GMC of Figure 2 is shown in Figure 3. A GMC whose associated CFG is such that the state X is the only terminal state and state X is reachable from every other state by some path is called properly terminating (PT) [GOST71]. Another related construction is the set of transformation expressions (TE's) of a GMC. Each TE describes a local

transformation in the token state due to the execution of a vertex. The TE's corresponding to Figure 2 are:

S -> A, B;
A -> C, M;
B, M -> D;
D -> F, R;
C, R -> E;
E, F -> X;

One way of determining if a GMC is properly terminating is by operating on the TE's with a reduction procedure [CERF72c]. The reduction procedure performs substitutions in the right hand side of the TE's and eliminates other TE's from the set of TE's. When the reduction procedure can operate no further, and if the only TE left in the set is "S -> X;" then the TE's and the GMC are called completely reducible (CR), and CR implies PT [CERF72c]. Figure 4 shows the reduction of the TE's given above. A good reference for more detail on the UCLA Graph Model is [GOST72].

SECTION 3 -- A Graph Model of A Communications Protocol

The protocol described in Section 2 is shown as a bigraph in Figure 5. Compare Figure 1 with Figure 5 and notice that the states in the former are arcs in the latter. Generally in a state machine a state represents "wait for something". In bigraphs this waiting condition is best represented by arcs. The meanings of the states in Figure 1 apply equally to the arcs in Figure 5.

A Initialize Sender
B Ready to send Message
C Wait for Acknowledgement or Time Out
D Good Acknowledgement Received
E Initialize Receiver
F Wait for Message
G Ready to Send Acknowledgement
H Check for Duplicate Message

The transformation expressions are derived from a bigraph by examining each vertex in turn. For example at the initial vertex the input arc is S and the output arcs are A and E, so the TE is S -> A, E. The set of TE's for this bigraph is:

```
(1) S -> A, E;
(2) A -> B;
(3) B -> C, MSG;
(5) C, ACK -> D;
(5) C, ACK -> C;
(6) D -> C;
(6) D -> B;
(4) C -> B;
(7) E -> F;
(8) F, MSG -> G;
(8) F, MSG -> F;
(9) G -> H, ACK;
(10) H -> F;
(10) H -> F;
(11) B, F -> X;
```

A portion of the corresponding computation flow graph is shown in Figure 6. This CFG is infinite due to the possibility of retransmission of the message an infinite number of times. This protocols must be modified to include a limitation on the number of retransmissions to some upper bound N.

A solution is presented in Figure 7. Note that arc L is initially issued N tokens. Each time a message is sent arc L loses one token but arc K gains one token so that the sum of the number of tokens on arcs L and K is always N, until an acknowledgement arrives. When the acknowledgement arrives it disables the loop and initiates collection of the tokens from arcs L and K. A bigraph cannot be properly terminating if there are tokens left in the graph which cannot be used. The segment in Figure 7 is called a counter-limited loop and collector.

It is important to note here the difficulty pointed out in Section 2, namely, that if the timing of retransmissions and acknowledgements is phased in a particular way a second copy of a message could be accepted as a new message. This difficulty will not be detected here as it arises from the lack of duplicate detection information, not from flaws in the control structure.

For simplicity in analysis the case is restricted to the situation where the range of sequence numbers is zero and one, used cyclicly and with the constraint that the communications system deliver messages in order. This second proviso allows modeling the communications system as a wire, or rather a pair of wires, one in each direction.

The implementation of duplicate detection by sequence numbers must be something like the following. For the sender of messages: in state N send message with sequence number equal to N and retransmit as necessary; when an acknowledgement arrives (correctly) if it carries sequence number N then go to state N + 1, if it carries some other sequence number discard it and remain in the same state. For the receiver of messages: in state M expect to receive a message; when a message arrives (correctly) send an acknowledgement carrying the same sequence number as in the message, if the sequence number in the message is M then go to state M + 1, if it is some other number remain in the same state. When limited to two states the arithmetic in the above is modulo two, thus the resulting values for M and N are zero and one.

In Figure 8 a portion of this mechanism is drawn. Arcs SS0, SS1, RS0, and RS1 represent the "states" of the sender and receiver; SS0 indicates that the sender is prepared to send a message with sequence number zero and to receive an acknowledgement with sequence number zero. Note that Figure 8 does not provide for discarding duplicate acknowledgements. In Section 2 it was noted that if the sequence number range was too close to the number of outstanding messages allowed and the communications network could deliver message out of order then errors could result. Here the assumption is made that messages arrive in the order sent, but this assumption must be included in the model.

Thus message sending and acknowledgement sending must be coordinated, this can be done by treating the communications facility as a resource and including arcs in the model to carry resource tokens representing the availability of that resource. In Figure 9 is a portion of such a construction. There are two arcs W1 and W2 representing the communications facility from sender to receiver (MSG path) and from receiver to sender (ACK path) respectively. Thus the communications network is modeled as two simplex paths each of which can hold exactly one message (acknowledgement).

Another aspect of this protocol that has not yet been discussed is the proper recovery when the retransmission loop is executed N times and no acknowledgement is received. The approach taken here is to move to a "quit" state and then reinitialize the procedure. A partial graph of this is shown in Figure 10.

To model correctly the communications protocol (of Figure 1) these ideas must be merged together in one graph. This results in the graph shown in Figure 11.

The protocol function of the various vertex groups in Figure 11 is described. Vertex 1 is the initiation vertex which starts the protocol in action. The group of vertices 3 through 13 is a counter-limited loop and collector. Of these vertices 6 and 7 are the basic loop, which transmits a message with sequence number zero up to N times. Vertex 8 leads to a quit state if the loop count is exhausted. Vertex 10 acts as a trigger when a good acknowledgement is received to start the collector. This whole group performs the protocol function of sending a message with sequence number zero up to N times, and upon receipt of a good acknowledgement stopping the transmission and changing the sender's state.

If a good acknowledgement is not received after N transmissions a quit state is activated. The vertex group 14, 15, and 16 receives messages with sequence number zero, sends acknowledgements, and when appropriate changes the receiver's state. Vertex 17 releases the transmission resource and enables the sending of messages with sequence number one. The vertices 18 through 26, 43, 44, and 45 collect either bad messages, bad acknowledgements, or good acknowledgements with the wrong sequence number and free the transmission resource while the sender or receiver remains in the same state. The vertex group 27 through 37 is structurally identical to, and performs the same function as, the group 3 through 13, except that this group sends messages with sequence number one. The vertex group 38, 39, and 40 is structurally identical to, and performs the same function as, the group 14, 15 and 16, except that this group receives messages with sequence number one.

Vertices 41 and 42 release the transmission resource and enable the sending of messages with sequence number zero. Vertices 46, 47, 48, and 49 gather together the various resources with the quit states and reinitialize the graph. This corresponds to a complete reinitialization of the protocol. Vertices 52 and 53 gather the resources in preparation for termination of the protocol. Vertices 50 and 51 represent a decision by the sender to stop. Vertices 54 and 55 represent a decision by the receiver to stop. Vertex 56 is the termination vertex of the graph and the protocol.

This is a very simple protocol yet to model it takes a fairly complex graph. This graph is properly terminating. The number of states in the computation flow graph with various values of N is shown in the following table. The CFG's were generated from the TE's by a computer program [CERF72a, CERF72b]. In addition to the increase in the number of states with the number of retransmissions (N), the

generation of CFG's becomes rapidly more difficult as the number of vertices and arcs increase.

N	STATES
1	75
2	395
3	555
4	723

Conclusion

These results further the belief that there is indeed some benefit in applying graph model techniques to protocol problems. The UCLA Graph Model property of proper termination can be determined in a mechanical and automatic way, and that this property indicates that a protocol is in a certain sense well behaved. However, it is clear that the practical protocols lead to fairly complex graphs. This motivates a search for ways of reducing the overall size and complexity of these graphs. This problem is attacked in [POST74].

As a result of this research it is strongly recommended that when developing protocols designers should at every step check their designs with graph modeling techniques.

When using the graph modeling technique to investigate a protocol, errors may arise at four levels. First, there may be a error in deriving the transformation expressions from the graph. Second, there may be an error in the mapping from the protocol to the graph. Third, there may be an error in interpreting the protocol. Fourth, there may be an error in the protocol itself.

A minor note is that the counter-limited loop and collector used in Section 4 does not appear to have been known before, and is a very useful construction when modeling any process which contains a computation step (in a loop) limited to a finite number of executions.

Graph Modeling of Computer Communications Protocols

Jonathan B. Postel

and

David J. Farber

Technical Report #85

May 1976

Jonathan B. Postel
Computer Science Department*
University of California
Los Angeles, California

David J. Farber
Department of Information and Computer Science
University of California
Irvine, California

*Current Address:

Augmentation Research Center
Stanford Research Institute
Menlo Park, California 94025

Introduction

The design of computer-to-computer communications protocols for networks of computers is one of the challenging problems facing the computer scientist. In the last few years, a number of large scale computer networks have been discussed and several of these are in various phases of implementation. It is clear that the design of the communications protocols for these networks is a difficult problem (though the evidence for this is mainly in internal memoranda, see [CARR70], [CROC72], [FARB73], [FRAN72], and [HEAR70]). Many of the problems in designing these communications protocols, which involve asynchronous parallel processes, are due to the difficulty in recognizing all of the possible orderings in which the significant events can occur and the possible execution sequences which arise from these orderings.

This paper draws from experience with the development of a computer network sponsored by the Advanced Research Projects Agency (ARPA) called the ARPANET [ROBE70], and the development of the UCLA Graph Model of Computation [ESTR63], especially the advances made by Gostelow [GOST71] and Cerf [CERF72c] which gave this model of parallel processing additional analytical power. One especially attractive aspect of this modeling technique is the existence of a computer program for testing for the property of proper termination. Proper termination indicates that the process modeled by the graph is in a certain sense well behaved; for example it has no deadlocks. These developments suggest that communications protocols can be modeled by the UCLA Graph Model in a useful and revealing way.

SECTION 1 -- A Communications Protocol

A communications protocol must deal with several sometimes contradictory issues. Among these are error control, flow control, multiplexing, and synchronization. The approaches taken to the issues have implications affecting retransmission frequency, buffering requirements, throughput, and delay. In contemplating all of these issues, two "zero level" issues must be kept in mind: deadlocks, and critical race conditions.

Deadlocks are situations in which two or more parties (processes, hosts) are waiting for a resource (command, signal) before proceeding (with their tasks) and that resource can only become available by one or more of those same parties proceeding. Critical race conditions are those situations where the outcome of a sequence of

(supposedly independent) operations is dependent upon the order in which the operations occur.

In the following section, the error control aspect of an example protocol is developed, then in Section 3 the example protocol is modeled.

A basic requirement at any level in a communications system is error control. Generally in the type of networks being considered the communications subsystems are designed to be very reliable. However, there may be interface hardware and some short transmission paths that are unchecked by the communications subsystem, also there may be temporary outages in the communications subsystem or messages lost due to congestion (and even errors which occur with very low probability can be costly); thus error checking is still required. A simple scheme for error control is to use a checksum attached to the message which the sender calculates and attaches to the message and the receiver calculates and compares with the checksum transmitted with the message. If there is a discrepancy, then an error has occurred. It is possible to use error correcting codes, but their discussion is beyond the scope of this presentation.

The sender must have some way of knowing that the message arrived at its destination correctly or incorrectly. A simple way to provide this knowledge is to have the receiver acknowledge each correctly received message. The sender can then assume that if no acknowledgement is received by the end of a timeout period then either the message was afflicted with errors or was undelivered. In this case the sender retransmits the message. Note that the acknowledgement must also be checksummed since it is transmitted via the same media as the message, and therefore subject to errors in the same way.

The protocol proposed thus far is not sufficient. It fails when the finite time for message delivery and the variability of that time is considered. If the delivery time in one instance is approximately equal to the timeout period, then the sender can decide to retransmit just as the receiver decides to send the acknowledgement (of the first copy). These then pass each other in the communications network. The result is that the receiver accepts two copies of the same message and the sender gets an extra acknowledgement. Further one or the other of the acknowledgements could be afflicted with errors and then the sender would not detect the problem either. There must be added to the protocol some means of identifying a message; for convenience, numbers will be used. Thus, if the sender attaches a sequence number to each message the receiver can

check to be sure it has not previously accepted a copy of that message. Because there must be some maximum sequence number, M, the sequence numbers will be used cyclicly, that is after sequence number M is used the sequence is restarted at zero. Another way to view this is to say that messages are counted indefinitely and that the sequence number is the message count modulo M+1. This protocol is illustrated in Figure 1.

In the program-like description below, if an "if clause" is false then fall through to the next statement (usually an if) at that same level (of indentation). If a "go to" directs the reexecution of a statement such as "if Ack received" then it is intended that a new instance of an Ack be received before the statement can be reexecuted.

The drawing is a state machine. In the notation accompanying an arc the statement above the bar indicates the input required to activate the arc, and the statement below the bar indicates the action taken while transiting the arc. A lambda indicates either no input or no action.

The sender starts in "initialization" state A, and moves to "send message" state B by setting N to zero (step S0). From state B the sender moves to "wait for Ack" state C by sending message N and starting a timer (S1, S2). In state C the arrival of an acknowledgement with a bad checksum (S3, S4, S5) results in continuing in state C. In state C the arrival of an acknowledgement with a good checksum (S3, S6) results in moving to "decision" state D. If the timer expires (S12, S13) the sender moves from state C to state B. If the sequence number of the received good acknowledgement is the same as the sequence number of the message sent then the sequence number is incremented and the sender moves from state D to state B (S7, S8, S9). If the received sequence number is not the same as the sent sequence number the sender moves from state D to state C (S10, S11).

The receiver begins in "initialization" state E, and moves to "wait for message" state F by setting M to zero (step R0). If the messages is received with a bad checksum the receiver remains in state F (R1, R2, R3). If the message is received with a good checksum the receiver moves to "send Ack" state G (R1, R4). The receiver moves from state G to "decision" state H by sending an acknowledgement (R5). If the sequence number received is not the expected one (R6, R7) the receiver moves to state F. If the sequence number is the expected one the receiver increments the sequence number and moves to state F (R8, R9, R10).

Sender	Receiver
S0: N=0	R0: M=0
S1: Send Msg(N)	R1: Rcv Msg(J)
S2: Start Timer	R2: if checksum = bad
S3: if Ack(I) received	R3: then go to R1
S4: if checksum = bad	R4: if checksum = good
S5: then go to S3	R5: Send Ack(J)
S6: if checksum = good	R6: if J not equal M
S7: if I = N	R7: then go to R1
S8: then N=N+1	R8: if J = M
S9: and go to S1	R9: then M=M+1
S10: if I not equal N	R10: and go to R1
S11: then go to S3	
S12: if Timer runout	
S13: then go to S1	

A Communications Protocol

A parameter to be considered in this mechanism is the number of different unacknowledged messages allowed. It has been shown that if this number is one, the sequence number need be only one bit [BART69]. However, when only one message is allowed to be outstanding the bandwidth of the communications network may be inefficiently used [METC73].

This example protocol while addressing only the issue of error control is sufficiently developed to lead to an interesting graph model and analysis.

SECTION 2 -- The UCLA Graph Model

The UCLA Graph Model has developed from the acyclic bilogic directed graph model introduced by Estrin [ESTR63]. The current model consists of arcs and vertices where generally speaking the vertices represent computations and the arcs represent the flow of control between computations. At each vertex there is either "exclusive or" (EOR) or "and" (AND) logic between the input arcs, and similarly EOR or AND logic between the output arcs. EOR input logic means that if any one of the input arcs is enabled (carries a token) the vertex can execute. AND input logic requires all input arcs to be enabled before the vertex can execute. EOR output logic indicates that when the vertex completes execution it will enable (place a token on) one of the output arcs. AND output logic indicates that when the vertex completes execution it will enable all the output arcs. A vertex executes by removing token(s) from its enabling input arc(s) and placing token(s) on its output

arc(s) as indicated by the output logic. If more than one of a set of EOR input arcs is enabled the vertex chooses randomly between the enabled arcs as to which token to remove. On EOR output logic, the vertex may decide which arc to enable based on an interpretation associated with the graph; in an uninterpreted graph the choice is random. The UCLA Graph Model also allows complex arcs, i.e. arcs with multiple tails or heads.

A UCLA Graph Model representation of a simple exchange of a message and reply is shown in Figure 2, where the vertices have the meanings:

1 = begin exchange
2 = send message
3 = receive message
4 = send reply
5 = receive reply
6 = end exchange

and the arcs have the meanings:

S = ready for exchange
A = ready to send message
B = ready to receive message
M = message available
C = read to receive reply
D = read to send reply
R = reply available
E = reply received
F = reply sent
X = exchange completed

A number of useful constructs and concepts have been associated with the UCLA Graph Model, the current development of which is called the Graph Model of Computation (GMC). Tokens are used in the UCLA Graph Model to analyze the flow of control through the graph as the vertices execute. A mechanism for testing GMC's for certain properties is called the Token Machine (TM). One of these constructs is the Computation Flow Graph (CFG) which is a state transition diagram where the states are named corresponding to the token placement in the GMC. The CFG for the GMC of Figure 2 is shown in Figure 3. A GMC whose associated CFG is such that the state X is the only terminal state and state X is reachable from every other state by some path is called properly terminating (PT) [GOST71]. Another related construction is the set of transformation expressions (TE's) of a GMC. Each TE describes a local

transformation in the token state due to the execution of a vertex. The TE's corresponding to Figure 2 are:

S -> A, B;
A -> C, M;
B, M -> D;
D -> F, R;
C, R -> E;
E, F -> X;

One way of determining if a GMC is properly terminating is by operating on the TE's with a reduction procedure [CERF72c]. The reduction procedure performs substitutions in the right hand side of the TE's and eliminates other TE's from the set of TE's. When the reduction procedure can operate no further, and if the only TE left in the set is "S -> X;" then the TE's and the GMC are called completely reducible (CR), and CR implies PT [CERF72c]. Figure 4 shows the reduction of the TE's given above. A good reference for more detail on the UCLA Graph Model is [GOST72].

SECTION 3 -- A Graph Model of A Communications Protocol

The protocol described in Section 2 is shown as a bigraph in Figure 5. Compare Figure 1 with Figure 5 and notice that the states in the former are arcs in the latter. Generally in a state machine a state represents "wait for something". In bigraphs this waiting condition is best represented by arcs. The meanings of the states in Figure 1 apply equally to the arcs in Figure 5.

A Initialize Sender
B Ready to send Message
C Wait for Acknowledgement or Time Out
D Good Acknowledgement Received
E Initialize Receiver
F Wait for Message
G Ready to Send Acknowledgement
H Check for Duplicate Message

The transformation expressions are derived from a bigraph by examining each vertex in turn. For example at the initial vertex the input arc is S and the output arcs are A and E, so the TE is S -> A, E. The set of TE's for this bigraph is:

```
(1) S -> A, E;
(2) A -> B;
(3) B -> C, MSG;
(5) C, ACK -> D;
(5) C, ACK -> C;
(6) D -> C;
(6) D -> B;
(4) C -> B;
(7) E -> F;
(8) F, MSG -> G;
(8) F, MSG -> F;
(9) G -> H, ACK;
(10) H -> F;
(10) H -> F;
(11) B, F -> X;
```

A portion of the corresponding computation flow graph is shown in Figure 6. This CFG is infinite due to the possibility of retransmission of the message an infinite number of times. This protocols must be modified to include a limitation on the number of retransmissions to some upper bound N.

A solution is presented in Figure 7. Note that arc L is initially issued N tokens. Each time a message is sent arc L loses one token but arc K gains one token so that the sum of the number of tokens on arcs L and K is always N, until an acknowledgement arrives. When the acknowledgement arrives it disables the loop and initiates collection of the tokens from arcs L and K. A bigraph cannot be properly terminating if there are tokens left in the graph which cannot be used. The segment in Figure 7 is called a counter-limited loop and collector.

It is important to note here the difficulty pointed out in Section 2, namely, that if the timing of retransmissions and acknowledgements is phased in a particular way a second copy of a message could be accepted as a new message. This difficulty will not be detected here as it arises from the lack of duplicate detection information, not from flaws in the control structure.

For simplicity in analysis the case is restricted to the situation where the range of sequence numbers is zero and one, used cyclicly and with the constraint that the communications system deliver messages in order. This second proviso allows modeling the communications system as a wire, or rather a pair of wires, one in each direction.

The implementation of duplicate detection by sequence numbers must be something like the following. For the sender of messages: in state N send message with sequence number equal to N and retransmit as necessary; when an acknowledgement arrives (correctly) if it carries sequence number N then go to state N + 1, if it carries some other sequence number discard it and remain in the same state. For the receiver of messages: in state M expect to receive a message; when a message arrives (correctly) send an acknowledgement carrying the same sequence number as in the message, if the sequence number in the message is M then go to state M + 1, if it is some other number remain in the same state. When limited to two states the arithmetic in the above is modulo two, thus the resulting values for M and N are zero and one.

In Figure 8 a portion of this mechanism is drawn. Arcs SS0, SS1, RS0, and RS1 represent the "states" of the sender and receiver; SS0 indicates that the sender is prepared to send a message with sequence number zero and to receive an acknowledgement with sequence number zero. Note that Figure 8 does not provide for discarding duplicate acknowledgements. In Section 2 it was noted that if the sequence number range was too close to the number of outstanding messages allowed and the communications network could deliver message out of order then errors could result. Here the assumption is made that messages arrive in the order sent, but this assumption must be included in the model.

Thus message sending and acknowledgement sending must be coordinated, this can be done by treating the communications facility as a resource and including arcs in the model to carry resource tokens representing the availability of that resource. In Figure 9 is a portion of such a construction. There are two arcs W1 and W2 representing the communications facility from sender to receiver (MSG path) and from receiver to sender (ACK path) respectively. Thus the communications network is modeled as two simplex paths each of which can hold exactly one message (acknowledgement).

Another aspect of this protocol that has not yet been discussed is the proper recovery when the retransmission loop is executed N times and no acknowledgement is received. The approach taken here is to move to a "quit" state and then reinitialize the procedure. A partial graph of this is shown in Figure 10.

To model correctly the communications protocol (of Figure 1) these ideas must be merged together in one graph. This results in the graph shown in Figure 11.

The protocol function of the various vertex groups in Figure 11 is described. Vertex 1 is the initiation vertex which starts the protocol in action. The group of vertices 3 through 13 is a counter-limited loop and collector. Of these vertices 6 and 7 are the basic loop, which transmits a message with sequence number zero up to N times. Vertex 8 leads to a quit state if the loop count is exhausted. Vertex 10 acts as a trigger when a good acknowledgement is received to start the collector. This whole group performs the protocol function of sending a message with sequence number zero up to N times, and upon receipt of a good acknowledgement stopping the transmission and changing the sender's state.

If a good acknowledgement is not received after N transmissions a quit state is activated. The vertex group 14, 15, and 16 receives messages with sequence number zero, sends acknowledgements, and when appropriate changes the receiver's state. Vertex 17 releases the transmission resource and enables the sending of messages with sequence number one. The vertices 18 through 26, 43, 44, and 45 collect either bad messages, bad acknowledgements, or good acknowledgements with the wrong sequence number and free the transmission resource while the sender or receiver remains in the same state. The vertex group 27 through 37 is structurally identical to, and performs the same function as, the group 3 through 13, except that this group sends messages with sequence number one. The vertex group 38, 39, and 40 is structurally identical to, and performs the same function as, the group 14, 15 and 16, except that this group receives messages with sequence number one.

Vertices 41 and 42 release the transmission resource and enable the sending of messages with sequence number zero. Vertices 46, 47, 48, and 49 gather together the various resources with the quit states and reinitialize the graph. This corresponds to a complete reinitialization of the protocol. Vertices 52 and 53 gather the resources in preparation for termination of the protocol. Vertices 50 and 51 represent a decision by the sender to stop. Vertices 54 and 55 represent a decision by the receiver to stop. Vertex 56 is the termination vertex of the graph and the protocol.

This is a very simple protocol yet to model it takes a fairly complex graph. This graph is properly terminating. The number of states in the computation flow graph with various values of N is shown in the following table. The CFG's were generated from the TE's by a computer program [CERF72a, CERF72b]. In addition to the increase in the number of states with the number of retransmissions (N), the

generation of CFG's becomes rapidly more difficult as the number of vertices and arcs increase.

N	STATES
1	75
2	395
3	555
4	723

Conclusion

These results further the belief that there is indeed some benefit in applying graph model techniques to protocol problems. The UCLA Graph Model property of proper termination can be determined in a mechanical and automatic way, and that this property indicates that a protocol is in a certain sense well behaved. However, it is clear that the practical protocols lead to fairly complex graphs. This motivates a search for ways of reducing the overall size and complexity of these graphs. This problem is attacked in [POST74].

As a result of this research it is strongly recommended that when developing protocols designers should at every step check their designs with graph modeling techniques.

When using the graph modeling technique to investigate a protocol, errors may arise at four levels. First, there may be a error in deriving the transformation expressions from the graph. Second, there may be an error in the mapping from the protocol to the graph. Third, there may be an error in interpreting the protocol. Fourth, there may be an error in the protocol itself.

A minor note is that the counter-limited loop and collector used in Section 4 does not appear to have been known before, and is a very useful construction when modeling any process which contains a computation step (in a loop) limited to a finite number of executions.

Bibliography

BART69 Bartlet, K.A., R.A. Scantlebury, and P.T. Wilkinson. "A Note on Reliable Full-Duplex Transmission over Half-Dulex Links," Communications of the ACM, 12(5):260-261, May 1969.

CARR70 Carr, C.S., S.D. Crocker and V.G. Cerf. "Host-Host Communications Protocol in the ARPA Network," AFIPS Conference Proceedings, 36:589-597, SJCC, 1970.

CERF71 Cerf, V., E.B. Fernandez, K.P. Gostelow, and S.A. Volansky. "Formal Control-Flow Properties of a Graph Model of Computation," Computer Science Department, ENG-7178, University of California, Los Angeles, December 1971 (UCLA-10P14-105).

CERF72a Cerf, V. and C. Maxwell. "An Automaton Generation Algorithm," Internal Memorandum 99, Digital Technology Research Group, Computer Science Department, University of California, Los Angeles, January 1972.

CERF72b Cerf, V. and C. Maxwell. "A Reduction Procedure," Internal Memorandum 108, Digital Technology Research Group, Computer Science Department, University of California, Los Angeles, February 1972.

CERF72c Cerf, V.G. Multiprocessors, Semaphores, and a Graph Model of Computation, Ph.D. Dissertation, ENG-7223, Computer Science Department, University of California, Los Angeles, April 1972 (UCLA-10P14-110).

CROC72 Crocker, S.D., J.F. Heafner, R.M. Metcalfe, and J.B. Postel. "Function-Oriented Protocols for the ARPA Computer Network," AFIPS Conference Proceedings, 40:271-279, SJCC, 1972.

ESTR63 Estrin, G. and R. Turn. "Automatic Assignment of Computations in a Variable Structure Computer System," IEEE Transactions on Computers, EC-12:756-773, December 1963.

FARB73 Farber, D.J. et. al. "The Distributed Computing System," Seventh Annual IEEE Computer Society International Conference, pp. 31-34, February 1973, (COMPCON 73).

FRAN72 Frank, H., R.E. Kahn, and L. Kleinrock. "Computer Communications Network Design--Experience with Theory and Practice," AFIPS Conference Proceedings, 40:225-270, SJCC, 1972.

GOST71 Gostelow, K.P. Flow of Control, Resource Allocation, and the Proper Termination of Programs, Ph.D. Dissertation, ENG-7179, Computer Science Department, University of California, Los Angeles, December 1971 (UCLA-10P14-106).

GOST72 Gostelow, K., V.G. Cerf, G. Estrin, and S. Volansky. "Proper Termination of Flow-of-Control in Programs Involving Concurrent Processes," Proceedings of the ACM, 25th Anniversary Conference, pp. 742-754, Boston, August 1972.

HEAR70 Heart, F.E., R.E. Kahn, S.M. Ornstein, W.R. Crowther, and D.C. Walden. "The Interface Message Processor of the ARPA Computer Network," AFIPS Conference Proceedings, 36:551-567, SJCC, 1970.

METC73 Metcalfe, R.M. Packet Communications, Ph.D. Thesis, Harvard University, Cambridge, Massachusetts, May 1973. Also available as Technical Report Number 114, Project MAC, Massachusetts Institute of Technology, Cambridge Massachusetts, August 1973.

POST74 Postel, J.B. A Graph Model Analysis of Computer Communications Protocols, Ph.D. Dissertation, Computer Science Depatrment, University of California, Los Angeles, March 1974.

ROBE70 Roberts, L.G. and B.D. Wessler. "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, 36:543-599, SJCC, 1970.

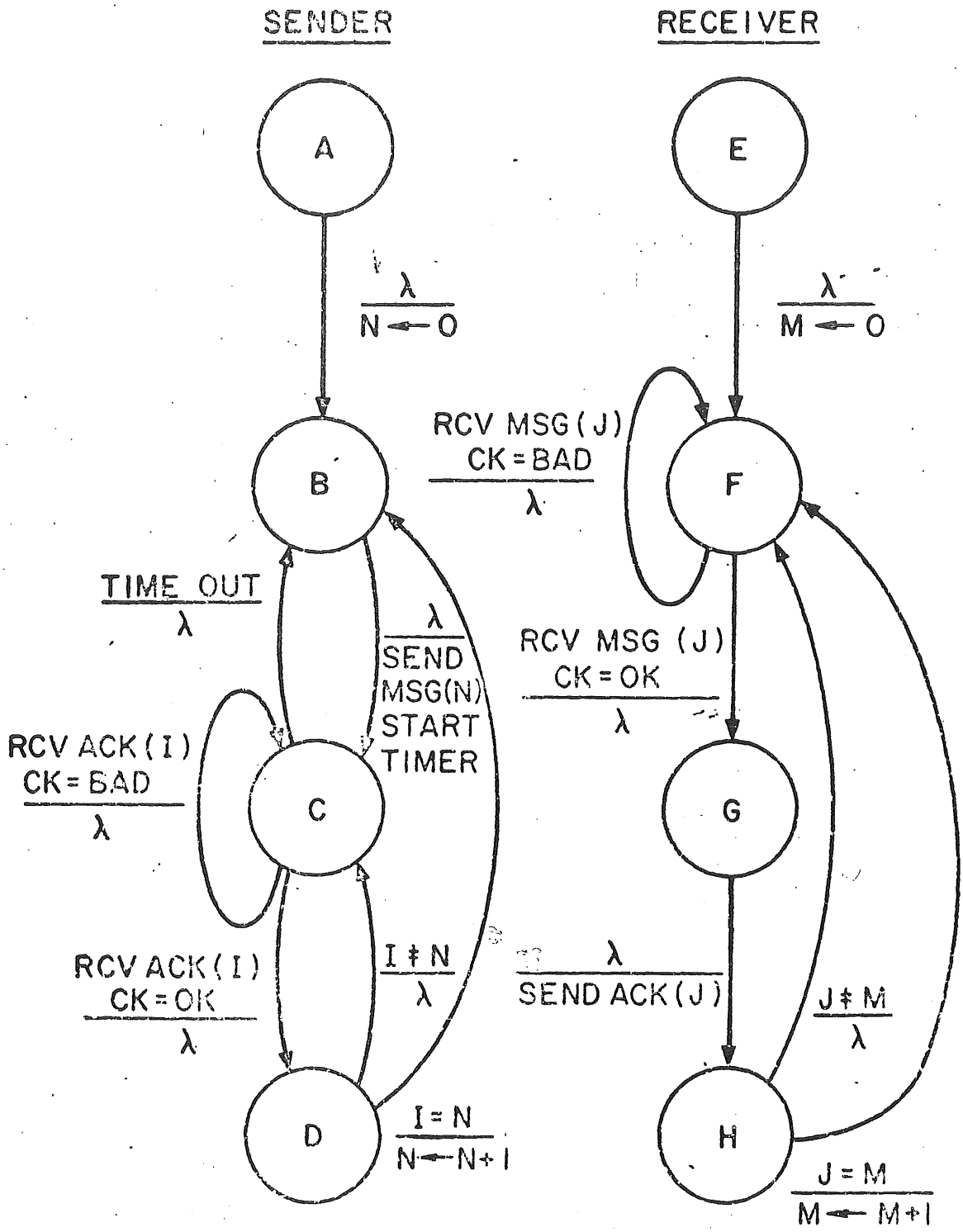

Figure 1 A Communications Protocol

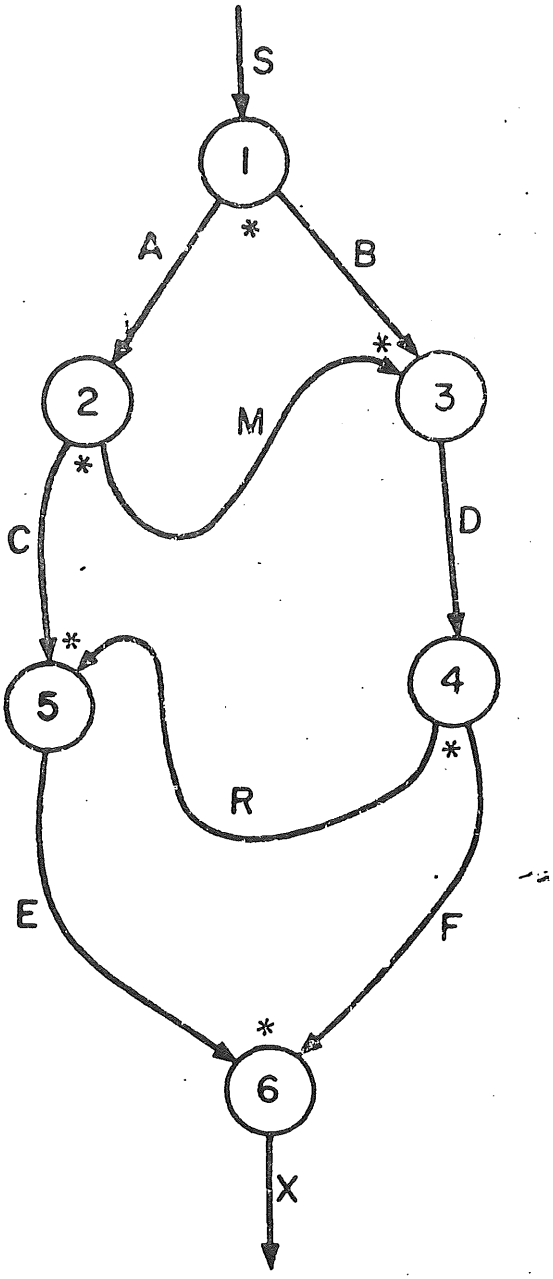

Figure 2 Send-Receive UCLA Graph Model

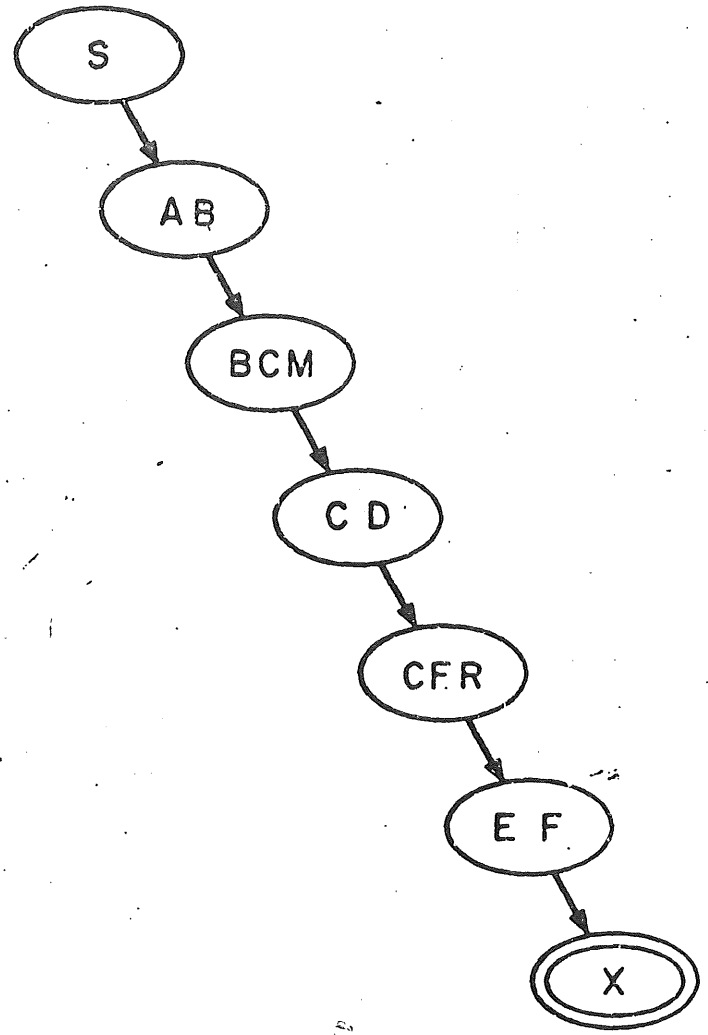

Figure 3 Send-Receive CFG

* S → A, B; * A → C, M; B, M → D; D → F, R; C, R → E; E, F → X;	* S → C, M, B; * B, M → D; D → F, R; C, R → E; E, F → X;	* S → C, D; * D → F, R; C, R → E; E, F → X;
0	1	2

* S → C, F, R; * C, R → E; E, F → X;	* S → E, F; * E, F → X;	S → X;
4	5	6

* Indicates TE's used in reduction step.

Figure 4 Send-Receive TE Reduction

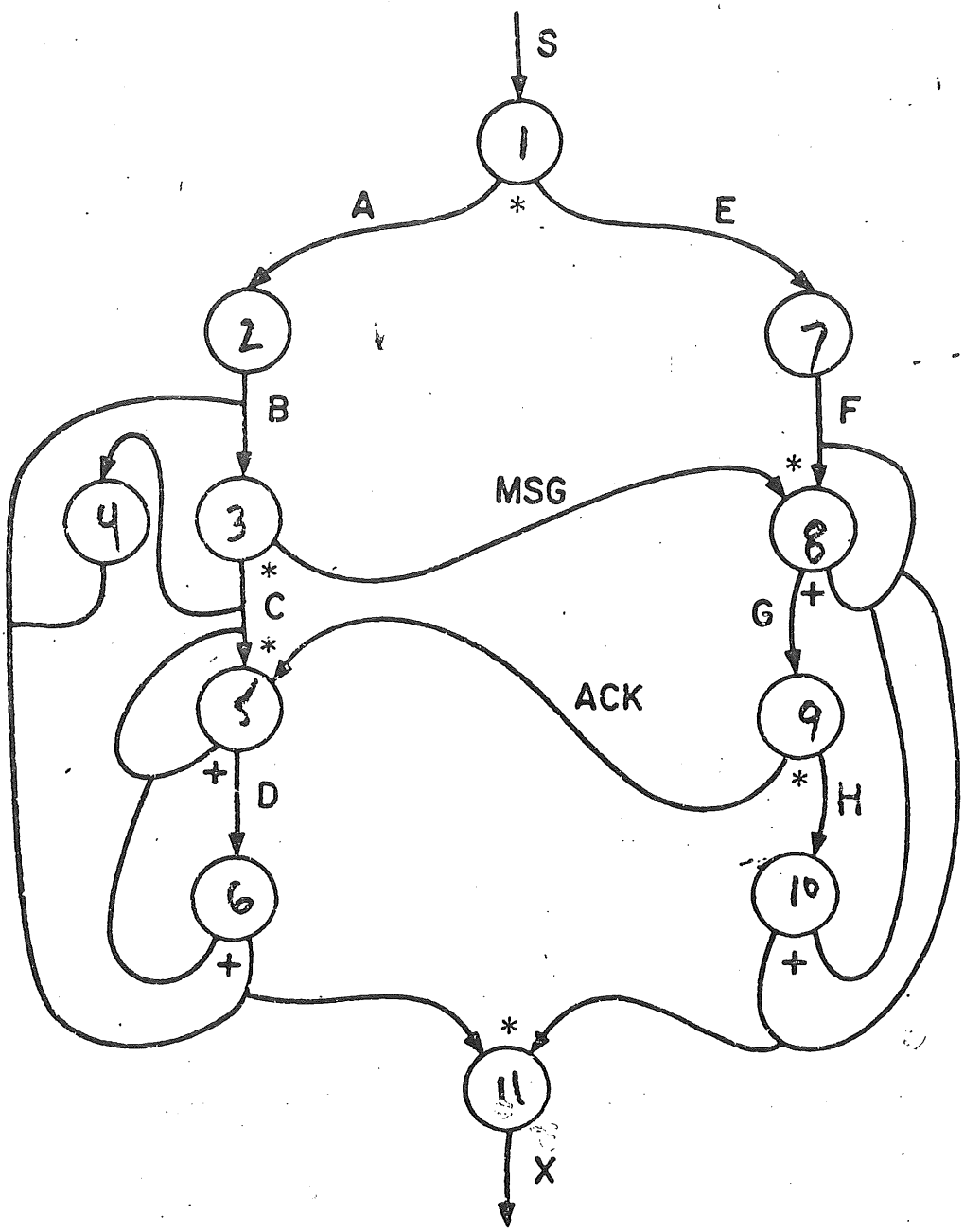

Figure 5 First Model of a Communications Protocol

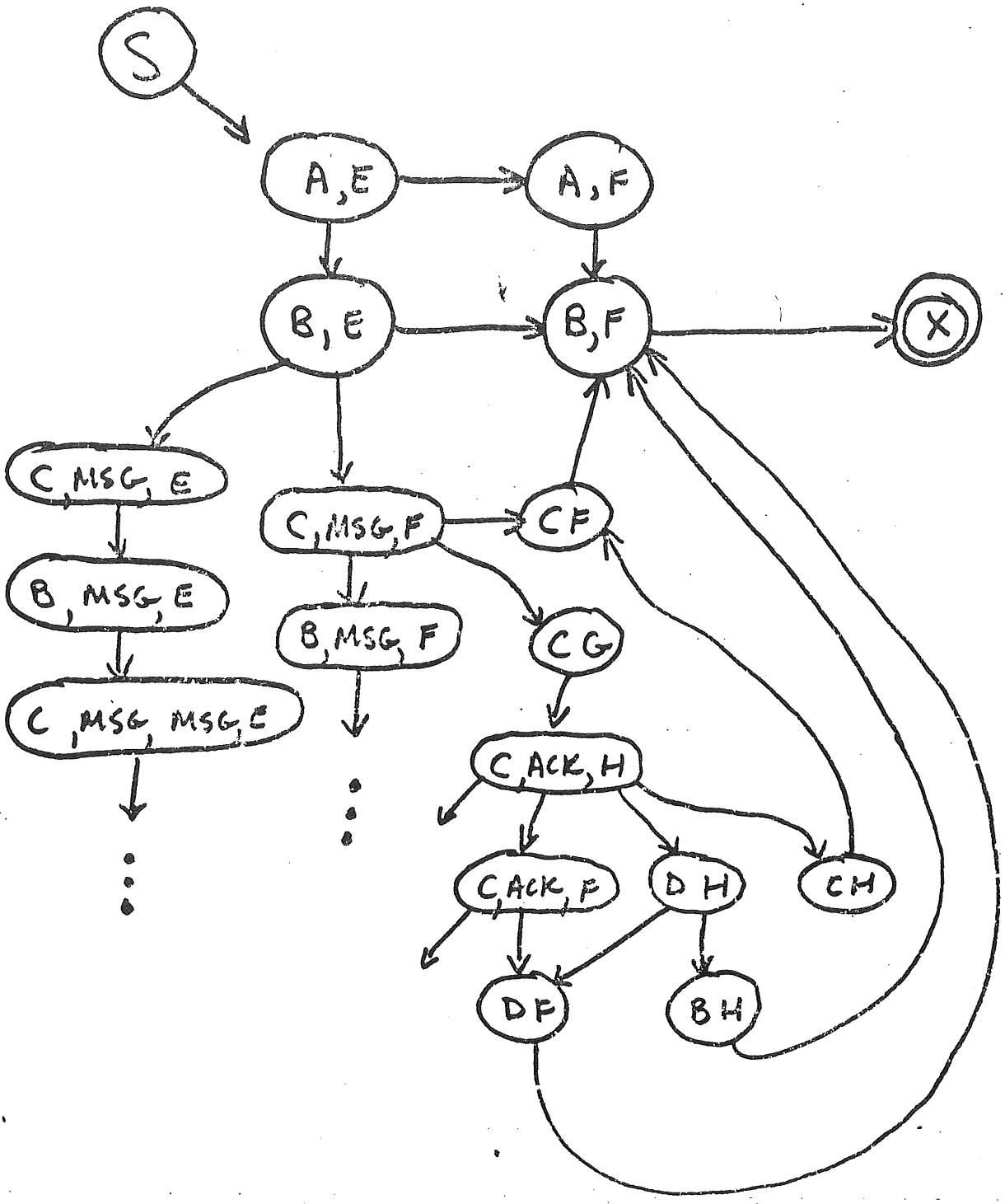

Figure 6 Partial CFG

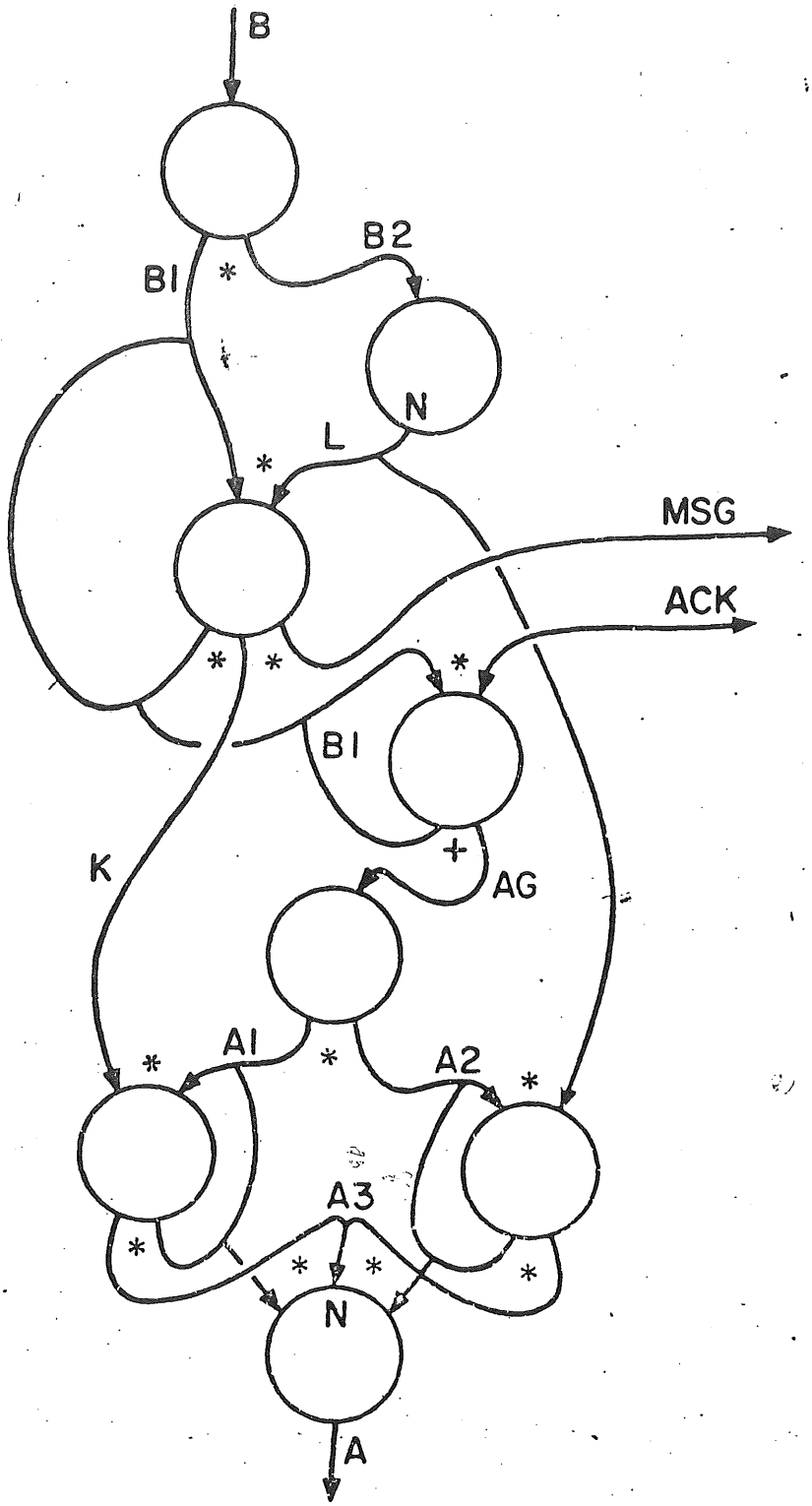

Figure 7 Counter Limited Loop and Collector

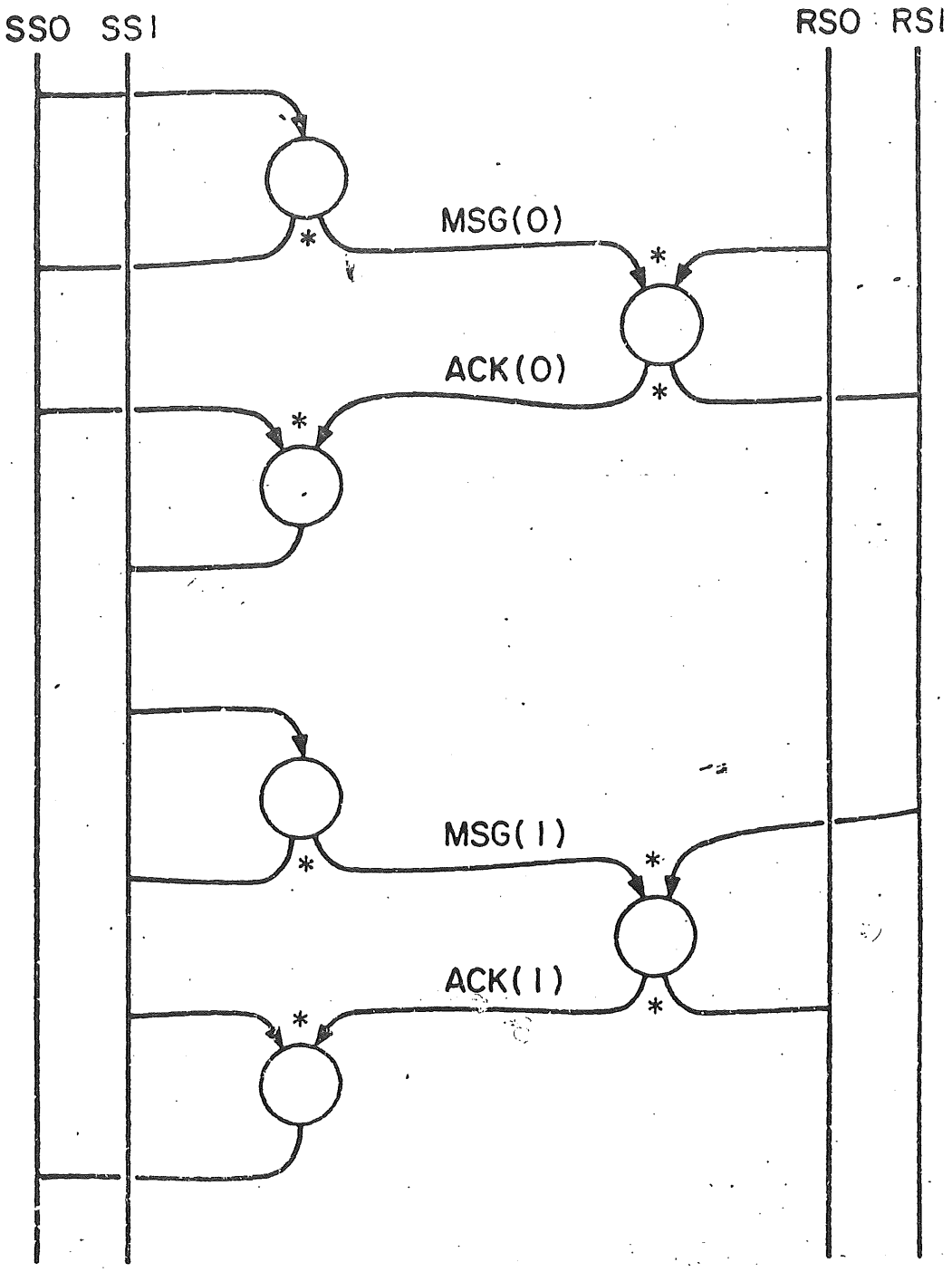

Figure 8 Duplicate Detection Transmission and States

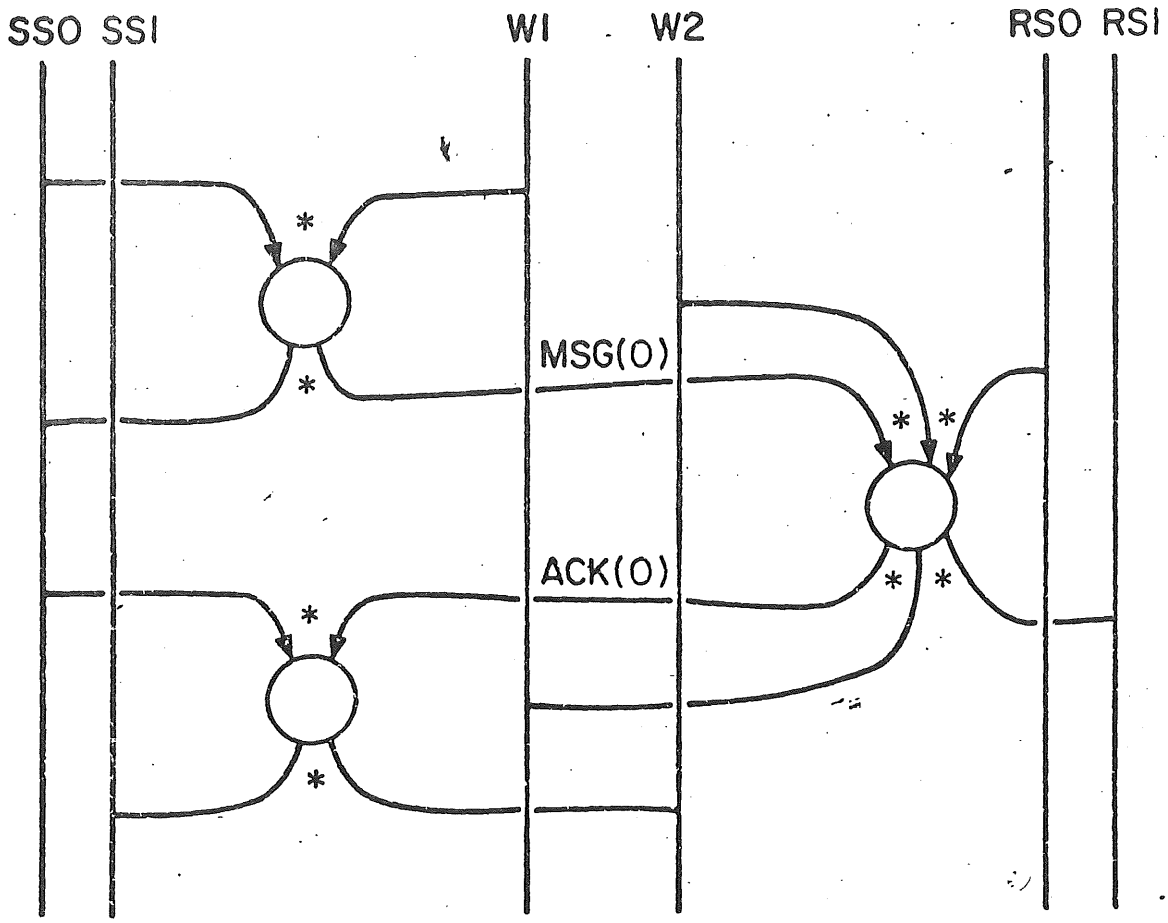

Figure 9 Transmission Resource Represented

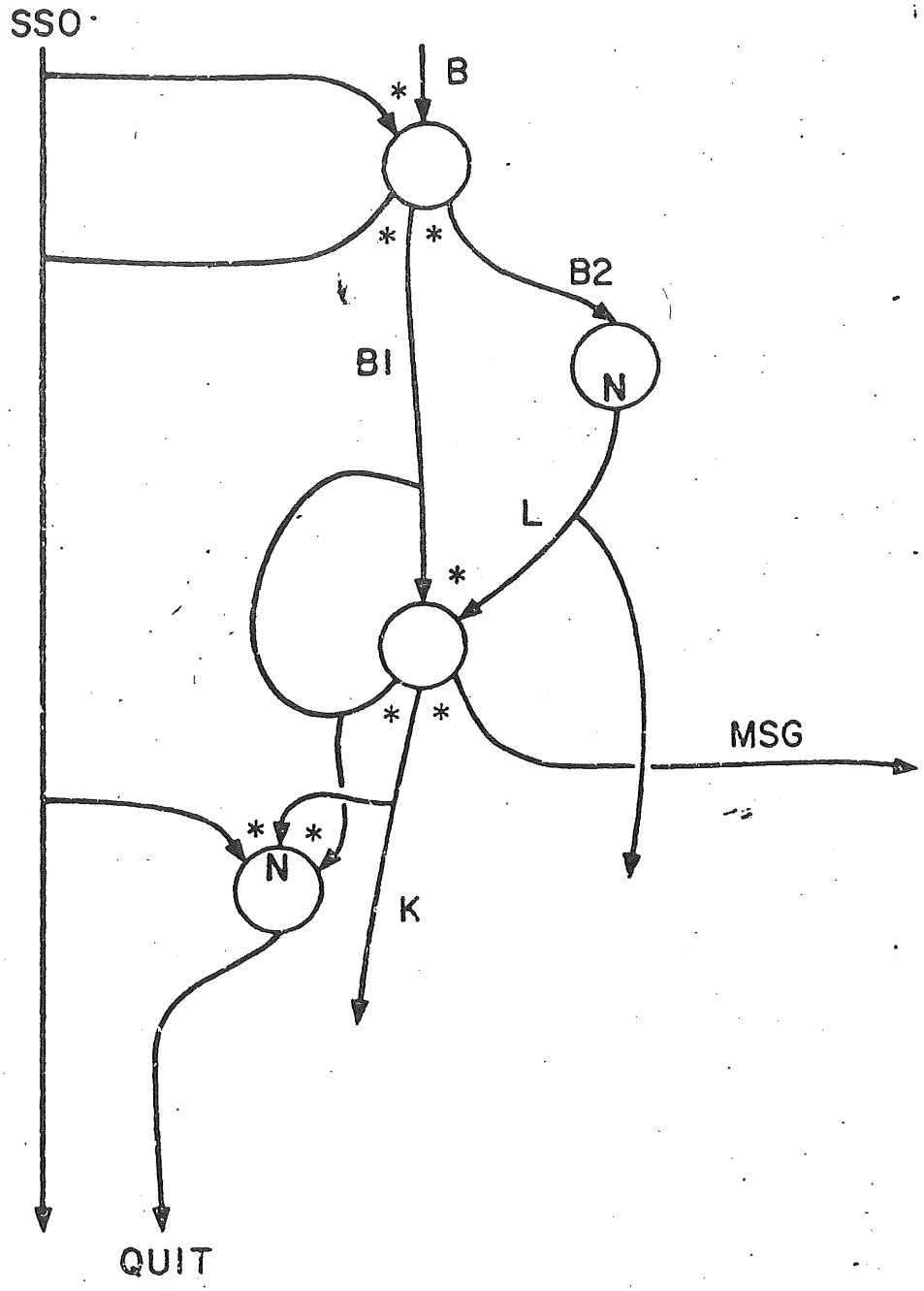

Figure 10 Bounded Retransmission with Quit State

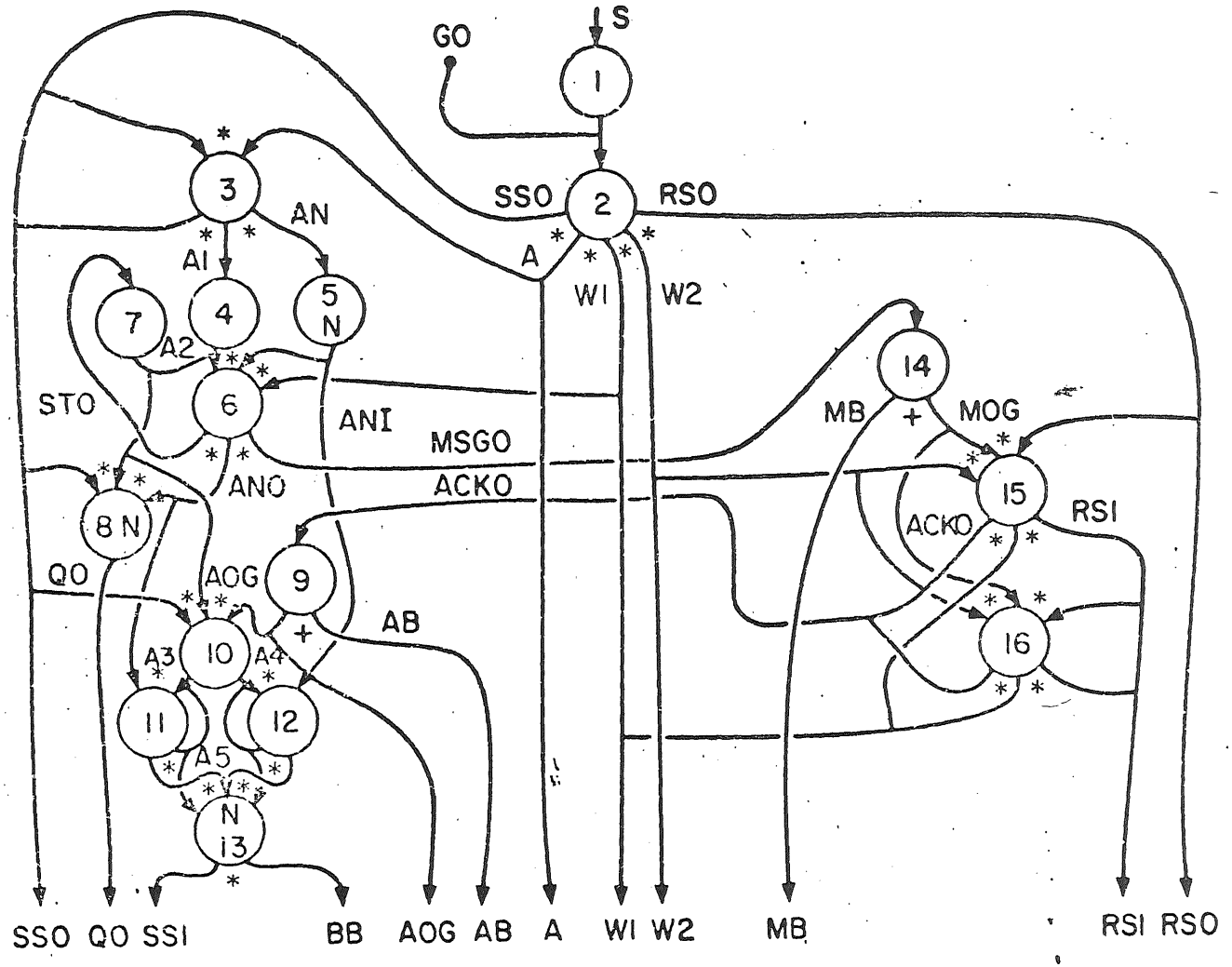

Figure 11 (Part 1) Graph Model of a Communications Protocol

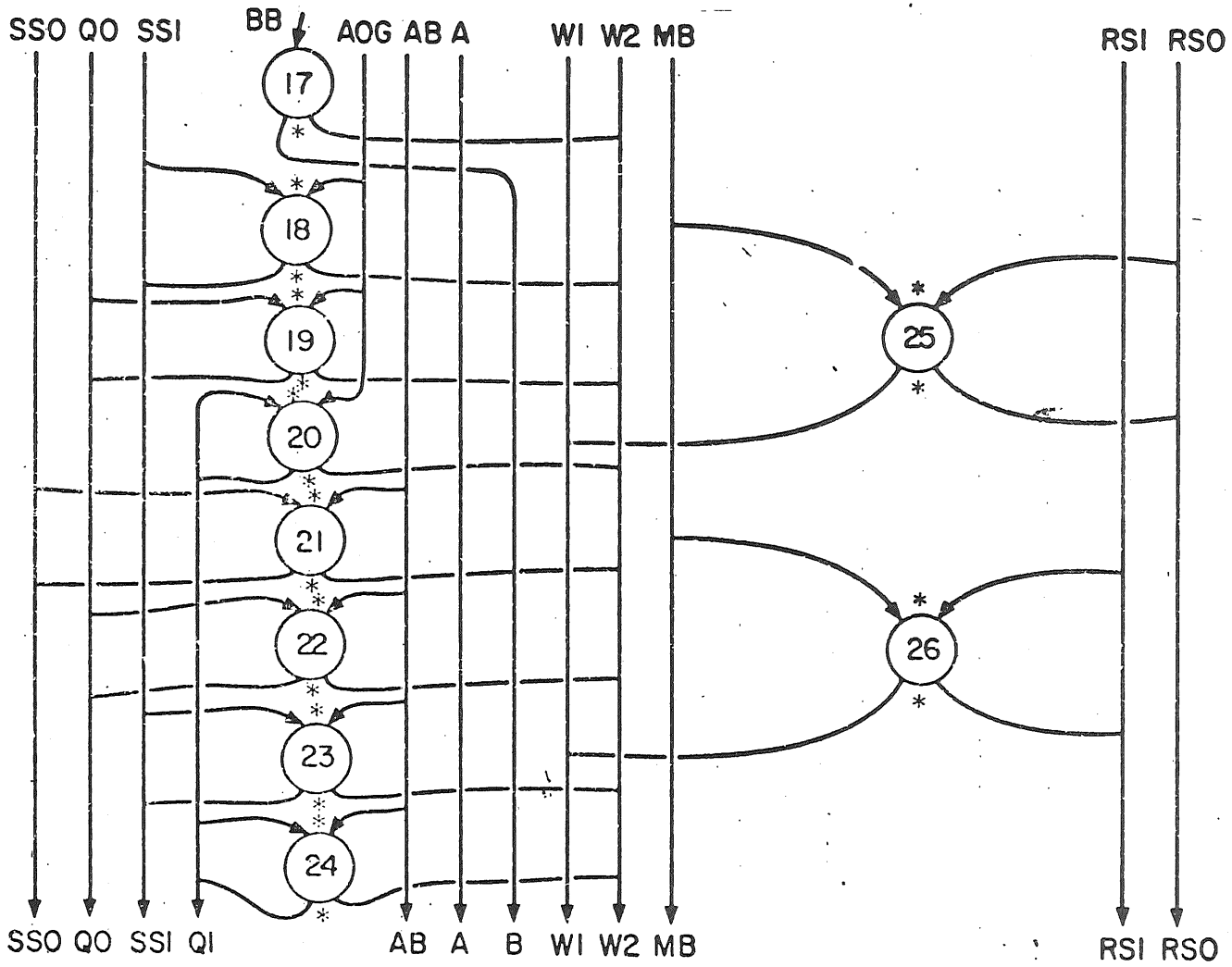

Figure 11 (Part 2) Garph Model of a Communications Protocol

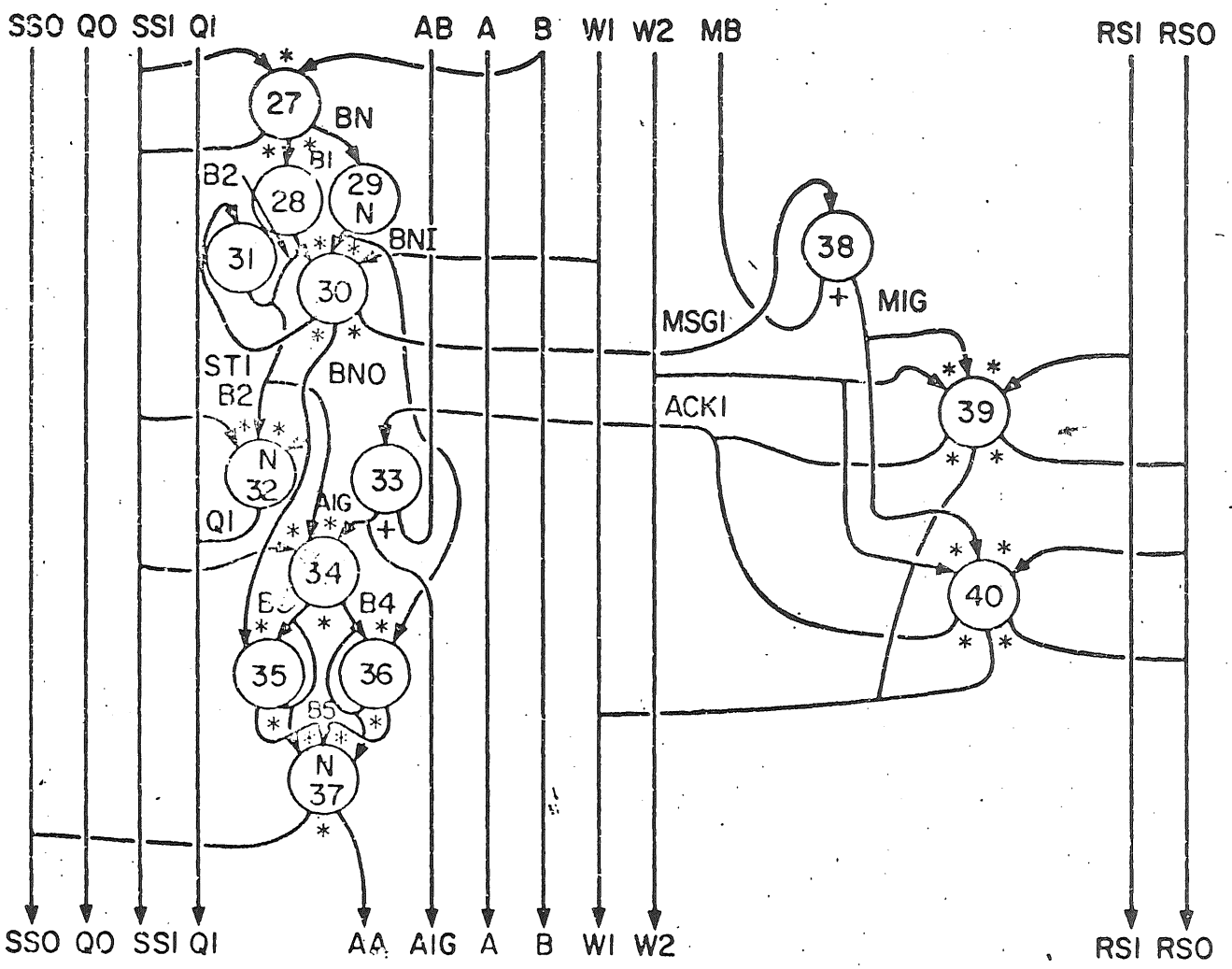

Figure 11 (Part 3) Graph Model of a Communications Protocol

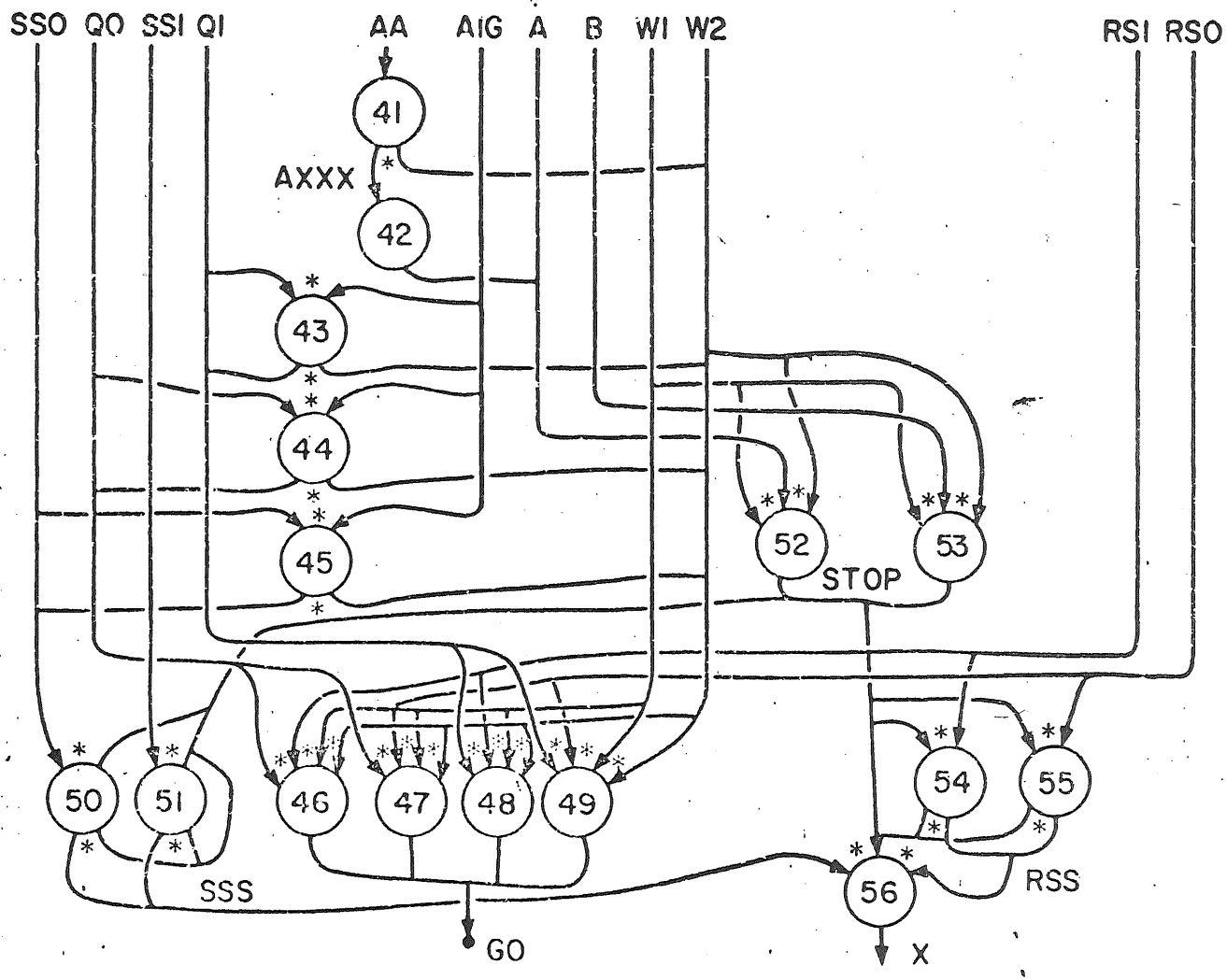

Figure 11 (Part 4) Graph Model of a Communications Protocol