

Single-Message Communication

DAG BELSNES

Abstract—When a communication system is used to transmit many short messages, it is important to reduce the amount of control overhead for creation and destruction of logical process-to-process connections and for reliable communication. Different end-to-end control procedures are described, and they are studied with respect to the possibility of losing a message or accepting a duplicate. It is shown (under certain assumptions about the communication network) that all end-to-end protocols either allow for loss of a message or can deliver duplicates of a message.

I. INTRODUCTION

IN SOME applications of data communication a great deal of the traffic consists of single-message conversations. That is, a source of a message wants to establish a connection to a given destination, send the message, and then immediately close the connection. Therefore, it is desirable to construct a procedure suitable for this type of use where the overhead in establishing and closing the connection is small.

In this paper we will discuss different ways to control the delivery of short messages through a data communication network. First some assumptions will be made about the communication system.

Suppose we have a process A in one host computer which periodically wants to transmit a single message to a process B in another computer. In order to be able to communicate, the host computers are interconnected by a data network (DN). The purpose of the data network is to deliver messages between the hosts, and it typically consists of transmission lines and switching machines. We will not be concerned about the internal design and operation of DN, but we will assume that DN is not perfect, such that a message given to DN may be lost or delayed unexpectedly long in DN.

Hence, in order to increase the reliability of message communication between processes each host will have a Network Control Program (NCP). Process A gives the data message to its NCP together with the address of B . The NCP's in the host of A (NCPA) and the host of B (NCPB) have the responsibility of controlling the delivery of the message to B .

Although DN is unreliable, the NCP's can increase the probability of correct communication between the processes by sending control messages in addition to the data messages. By correct communication we mean that either B accepts one and only one copy of the message or that B does not get the message, and that A in both cases is informed about the result.

A message is said to be lost if A thinks that B has received

Paper approved by the Associate Editor for Computer Communication of the IEEE Communications Society for publication without oral presentation. Manuscript received June 5, 1975; revised September 12, 1975. The first draft of this paper was prepared during a stay at Stanford University, Stanford, CA, and supported by Norges Teknisk-Naturvitenskapelig Forskningsrad.

The author is with the Computer Center, University of Oslo, Oslo, Norway.

it but this is not the case. A message is said to be duplicated if B receives two copies of it but thinks they are different messages.

We assume that the number of possible pairs of processes that may want to communicate is so large that the NCP can not keep any information about inactive connections. Further we assume that a NCP may lose control information about active connections (for instance, after a reload of a front-end computer containing the NCP).

II. RELIABILITY OF INTERPROCESS COMMUNICATION

A major problem with single-message process communication is to define a protocol between the NCP's that avoids loss or duplicates of a message. It is also important to have little overhead in establishing and closing single-message conversation.

However, with the assumptions made above it is impossible, regardless of how elaborate we make the NCP protocol, to achieve both that no message from A to B gets lost, and that no message from A to B gets duplicated.

The following is a justification of this statement.

When a process A transmits a message to another process B , the following three events occur in a fixed sequence at different times.

- 1) A gives the message to the local NCP.
- 2) B receives the message from its local NCP.
- 3) A is notified by the NCP that B has accepted the message.

Assume that the connection between A and B is broken at a moment after event 1) but before event 3) in such a way that the network, including the NCP's, has no information left about the connection. In this case A can not find out whether event 2) has occurred or not. A can decide to give the message once more to the NCP and perhaps introduce a duplicate, or A can assume that B has received the message and not send it again in which case the message may be lost [if event 2) has not occurred].

Note that the possibility of either losing data or accepting duplicates applies for all types of communication protocols. It is also independent of the design of the subnet (packet switched, circuit switched, ...). It is only based upon having a communication system between A and B that may break with loss of status information about the connection.

This loss/duplicate observation implies that either loss of messages must be tolerated, or duplicates have to be accepted. In most process-to-process communication presumably fewer problems arise from duplication of a message than from loss of a message. Although some applications may work with loss of messages, we will in the following try to make the communication protocol rule out the possibility of losing messages, and rather allow for duplicates. If it is important for an application not to accept a duplicate, this can be accomplished on user

level by including some information in the message to detect the duplicate.

We will now try different control procedures for single-message communication and look at the possibilities for losing the message or accepting a duplicate. In the rest of the paper our basic model of DN is a packet or message switching system. All control procedures except the five packet version have deficiencies when the communication system between *A* and *B* works without any breaks. In order to describe the packet interchange we will use the same notation as in [1] and [3].

A packet in the net will contain the following information which is of interest for our discussion. In order to send long data messages (using more than one packet) and to have a reference for the data acknowledgments messages, the data characters are to be considered as numbered. Thus, each packet contains a sequence number which identifies the previous data character sent in the same direction.

The packets may also contain some control information.

- syn First packet of a connection.
- ack *n* Acknowledgment of all data characters including character number *n* received in the opposite direction.
- rel No more data will be sent in this direction for the given connection.

Each line in the description of the dialog consists of a packet label in parentheses followed by the activity at NCPA where “←” signifies the packet being received by NCPA, “→” signifies the packet being transmitted by NCPA, and “...” signifies that NCPA has not yet received the packet or that NCPA has transmitted the packet earlier. Next appears a description of the packet in the form ⟨seq *n*⟩ ⟨control info⟩ ⟨data⟩. Next appears the activity at NCPB with a corresponding interpretation as for NCPA. When a packet is retransmitted it will be given a new label. When the delay between transmission and reception of a packet is unimportant, the packet will be shown to be transmitted and received on the same line.

III. ONE-PACKET INTERCHANGE

The simplest NCP control procedure (no control at all) is to just send the data message and then close the connection. That is, a protocol without any acknowledgments or retransmissions. Obviously this procedure will lose a message each time DN fails to deliver the message. There is no possibility of introducing a duplicate.

IV. TWO-PACKET INTERCHANGE

A natural way to avoid the type of loss mentioned above is to let NCPA require an acknowledgment from NCPB before NCPA reports success to process *A*. The NCP protocol is as follows.

When *A* gives a message to NCPA, NCPA opens a connection to *B* and transmits the data message to NCPB. By opening a connection we mean that the NCP sets up a *transmission control block (TCB)* which keeps control information about the connection as long as the connection is active. By closing a connection we mean that the NCP removes the TCB.

When NCPB receives the message, then NCPB will open the connection, give the data to *B*, send an acknowledgment message back to NCPA, and close the connection. NCPA will, upon receiving the acknowledgment, inform *A* and then close the connection. If NCPA does not receive the acknowledgment within a given time limit, NCPA retransmits the data message. Thus, if the data message is lost in DN, the retransmission scheme is meant to ensure that *B* sooner or later will receive the data.

A normal conversation looks as follows:

	NCPA	NCPB
(M1)	→ ⟨seq ∅⟩ ⟨syn,rel⟩ ⟨15 data bytes⟩	→
(M2)	← ⟨seq ∅⟩ ⟨syn,rel,ack 15⟩	←

However, this procedure may easily introduce duplicates. If NCPA has a timeout, it may be because M1 is lost (delayed) in DN, or because M2 is lost (delayed) in DN. If the latter is the case, NCPB will receive the same data when NCPA retransmits. Since NCPB closed the connection when M2 was sent, NCPB has no way of detecting the duplicate.

More surprising is the fact that this protocol also allows for loss of messages. Consider the following dialog:

	NCPA	NCPB
(M1)	→ ⟨seq ∅⟩ ⟨syn,rel⟩ ⟨15 data bytes⟩	→
(M2)	... ⟨seq ∅⟩ ⟨syn,rel,ack 15⟩	←
	<i>B</i> accepts data and NCPB closes the connection	
(M1*)	→ ⟨seq ∅⟩ ⟨syn,rel⟩ ⟨15 data bytes⟩	...
	NCPA retransmits after a timeout	
(M2)	← ⟨seq ∅⟩ ⟨syn,rel,ack 15⟩	...
	late arrival of M2. NCPA closes the connection and <i>A</i> is informed that <i>B</i> has accepted the data	
(M1*)	... ⟨seq ∅⟩ ⟨syn,rel⟩ ⟨15 data bytes⟩	→
	late arrival of M1*	
(M3)	... ⟨seq ∅⟩ ⟨syn,rel,ack 15⟩	←
	<i>B</i> accepts duplicate, NCPB closes the connection	
(M4)	→ ⟨seq ∅⟩ ⟨syn,rel⟩ ⟨15 new data bytes⟩	lost
(M3)	← ⟨seq ∅⟩ ⟨syn,rel,ack 15⟩	...
	NCPA closes the connection and <i>A</i> is informed that <i>B</i> has accepted M4. M4 is lost.	

The problem arises because an acknowledgment (M3) from an old connection is accepted as a legal packet in the new connection. The confusion can be avoided in different ways under certain assumptions.

Assume that there exists a maximum time *T* that packets can live in the network. If the network design is such that a message is very seldom destroyed in the network (retransmissions between adjacent switching nodes), a reasonable solution will be to have a timeout for retransmission from the source that is greater than $2*T$. Another way is to require a time delay of at least $2*T$ between new connections between the same processes. However, these proposals are not satisfactory for networks where *T* is very large.

With any of these requirements we can prove that the two-packet protocol will not lose a data message from *A* to *B*.

Long timeout between retransmissions: When *NCPA* sends a data message, no old acknowledgment can be received by *NCPA*. If the data message is retransmission, this is true because of the long timeout. If the data message is a new one, it is true because *NCPA* has received the acknowledgment for the previous data message. This means that an acknowledgment received by *NCPA* must apply to the current data message.

Long delay between new connections: Caused by rapid retransmissions, *NCPA* can receive many acknowledgments, but they will all refer to the last data message. When *NCPA* sends a new data message, no old acknowledgment can be received by *NCPA* because of the long delay before the new data message was sent.

V. THREE-PACKET INTERCHANGE

The two-packet protocol may introduce a duplicate each time an acknowledgment message (M2) is lost (delayed) in DN. In order to control the reception of M2 we can include a dummy data byte in M2 and require that *NCPA* has to send an acknowledgment to M2. Thus, the conversation looks as follows:

NCPA	NCPB
(M1) → ⟨seq 0⟩ ⟨syn,rel⟩ ⟨10 data bytes⟩	→
NCPA and NCPB have opened the connection	
(M2) ← ⟨seq 0⟩ ⟨syn,rel,ack 10⟩ ⟨1 dummy⟩	←
<i>B</i> accepts the data	
(M3) → ⟨seq 10⟩ ⟨ack 1⟩	...
NCPA informs <i>A</i> about success and removes the TCB (closes)	
(M3) ... ⟨seq 10⟩ ⟨ack 1⟩	→
NCPB removes the TCB.	

In this case, if M2 is lost the connection is still open, both in *NCPA* and *NCPB*, such that a retransmission of M1 or M2 will resolve the situation. However, this scheme also allows for both loss of a message and occurrences of duplicates. Loss of a message can occur in the following way due to confusion between different connections between *A* and *B*.

NCPA	NCPB
(M1) → ⟨seq 0⟩ ⟨syn,rel⟩ ⟨10 data bytes⟩	→
(M2) ← ⟨seq 0⟩ ⟨syn,rel,ack 10⟩ ⟨1 dummy⟩	←
<i>B</i> accepts the data and sends an acknowledgment	
(M3) → ⟨seq 10⟩ ⟨ack 1⟩	lost
NCPA removes the TCB	
(M1*) → ⟨seq 0⟩ ⟨syn,rel⟩ ⟨10 new data bytes⟩	lost
(M2*) ← ⟨seq 0⟩ ⟨syn,rel,ack 10⟩ ⟨1 dummy⟩	←
NCPB retransmits M2	
(M3*) → ⟨seq 10⟩ ⟨ack 1⟩	→
NCPA and NCPB remove the TCB's.	
No one knows that M1* is lost.	

In this case it does not help with long delays between retransmissions or different connections. Even if it is a long time between M3 and M1*, *NCPB* can consume this time by repeated retransmissions of M2.

A solution is to have a clever way of choosing the initial sequence number on a connection, such that old packets in the net will have improper sequence numbers. Here we have implicitly assumed that the range of sequence numbers is so large that a new and an old packet in the net can not contain the same sequence number. References [3], [4] suggest using a real-time clock in the host to select the initial sequence number. This also works if the host has a temporary breakdown where all information about the connection is lost.

Then M1* will have a different sequence number than 0, say *x*. When *NCPA* receives M2*, *NCPA* discovers that this is an old message because the number in the ack-field is out of range. Then the possibility of losing a message is ruled out, assuming that a retransmission sequence does not last so long that the initial sequence number goes through a complete cycle.

Note that the technique with clever choice of initial sequence number also resolves the problem with the two-packet protocol.

The three-packet protocol still has problems with duplicates even if there is no break in the connection (the NCP's are working).

NCPA	NCPB
(M1) → ⟨seq <i>x</i> ⟩ ⟨syn,rel⟩ ⟨21 data bytes⟩	→
(M2) ... ⟨seq <i>y</i> ⟩ ⟨syn,rel,ack <i>x</i> +21⟩ ⟨1 dummy⟩	←
Data delivered to <i>B</i>	
(M1*) → ⟨seq <i>x</i> ⟩ ⟨syn,rel⟩ ⟨21 data bytes⟩	...
NCPA retransmits after a timeout	
(M2) ← ⟨seq <i>y</i> ⟩ ⟨syn,rel,ack <i>x</i> +21⟩ ⟨1 dummy⟩	...
Late arrival of M2	
(M3) → ⟨seq <i>x</i> +21⟩ ⟨ack <i>y</i> +1⟩	→
NCPA and NCPB release TCB	
(M1*) ... ⟨seq <i>x</i> ⟩ ⟨syn,rel⟩ ⟨21 data bytes⟩	→
Late arrival of M1*. NCPB thinks it is a new connection	
(M2*) ← ⟨seq <i>y'</i> ⟩ ⟨syn,rel,ack <i>x</i> +21⟩ ⟨1 dummy⟩	←
(M3*) → ⟨seq <i>x</i> +21⟩ ⟨ack <i>y'+1</i> ⟩ ⟨error no connection⟩	→
NCPA has no TCB and so informs NCPB	

However, *NCPB* must in this case accept the duplicate, because otherwise a message may be lost. (In the case M3 is lost/delayed and *NCPB* retransmits M2. *NCPB* will then also receive an M3*-type message as reply.)

Reference [3] suggests the following modification to the three-packet procedure. *B* does not accept the data at once [when *NCPB* returns the acknowledgment (M2)], but waits until M3 arrives. However, this makes the meaning of an ack unclear (does not, any longer, necessarily mean that data are given to the user), and the problem of loss/duplicate still exists. Assume that *NCPB* does not receive M3. After a timeout *NCPB* will retransmit M2. Since *NCPB* closed the connection when *NCPA* transmitted M3, *NCPB* will receive the reply

TABLE I
RELIABILITY OF THE DIFFERENT PROTOCOLS

	without NCP breaks		with NCP breaks	
	lose data	accept dupl.	lose data	accept dupl.
one packet protocol	yes	no	yes	no
two packet protocol	no	yes	no	yes
three packet protocol	no	yes	no	yes
four packet (ver. i)	no	yes	no	yes
four packet (ver. ii)	no	no	yes	no
five packet protocol	no	no	no	yes

since we have to live with duplicates anyway, these should be removed on the user level.

VIII. SUMMARY

In all interprocess communication using an unreliable transmission media, it is impossible to prevent both loss of messages and acceptance of duplicates. In order to avoid undetected loss of a message it is necessary to differentiate messages from different connections. This can be done by choosing the initial sequence number for the packets as described in [3], [4].

Then two- to five-packet protocols can be defined such that they never lose messages even if a host has a restart. For single-message communication it is necessary with a five-packet interchange, if we want to insure that delivery of a duplicate message to the user can only occur in connection with a host breakdown. With two-, three-, and four-packet interchanges, duplicates may be given to the user when packets are lost or delayed in the net (DN).

The four-packet interchange can be redefined such that it will not lose messages or create duplicates as long as the hosts are working. However, this scheme may lose a message during a host restart.

Since the possibility of duplicates cannot be ruled out if we want to prevent losses, and since we want little overhead for single message communication, the two-packet protocol may be the best choice. Anyway, the user should provide means for duplicate detection at the user level, if acceptance of duplicates is disastrous.

Table I contains a summary of the results for the different protocols.

ACKNOWLEDGMENT

The author wishes to thank V. G. Cerf, Y. Dalal, and C. A. Sunshine for many fruitful discussions about problems in network protocol design and for preparing the paper.

REFERENCES

- [1] V. G. Cerf, Y. Dalal, and C. A. Sunshine, "Specification of internet transmission control program," Stanford Univ., Stanford, CA, INGW-note 72, Dec. 1974.
- [2] V. G. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. COM-20, pp. 637-648, May 1974.
- [3] R. Tomlinson, "Selecting sequence numbers," BBN, INGW protocol note 2, Aug. 1974.
- [4] Y. Dalal, "More on selecting sequence numbers," Stanford Univ., Stanford, CA, INGW protocol note 4, Oct. 1974.
- [5] D. Walden, "A system for interprocess communication in a resource sharing network," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 221-230, Apr. 1972.
- [6] C. Sunshine, "Issues in communication protocol design—Formal correctness," Stanford Univ., Stanford, CA, INGW protocol note 5, Oct. 1974.



Dag Belsnes was born in Oslo, Norway, on August 26, 1941. He received a degree in mathematics (comparable to a M.S.) from the University of Oslo, Oslo, Norway, in 1967.

Since 1968 he has been an Assistant Professor at the Computer Centre of the University of Oslo, where he has worked with the implementation of a SIMULA compiler for the CDC 3300 and CDC 3600 computers. Currently he is responsible for the development of a local computer network at the University.

Mr. Belsnes is a member of the Association for Computing Machinery and Norsk Selskap for Elektronisk Informasjonsbehandling.