bcc	title		prefix/class-number.revision
	AN ERROR-FREE COMMUNICATIONS LINE		FAROUT/W-40

checked [illegible]	authors	approval date 1/6/70	revision date
checked Larry L Barnes	Paul Heckel	classification Working Paper	
approved Mel	[illegible]	distribution Company Private	pages 30

ABSTRACT and CONTENTS

This document defines algorithms designed to make an error prone 4800-baud communications line (almost) error free. The error-free communications system that results is described and analyzed.

TABLE OF CONTENTS

INTRODUCTION

Transmission lines, like humans, make errors and therefore cause much anguish. With both, one is faced with the problem of devising a system with enough redundancy to detect, and in some cases correct the errors. With both, one would like to do this while minimizing the loss in efficiency.

The communications line in question is a 4800-baud conditioned phone line that will be used to connect the M-1 CHIO to a Remote Concentrator. From time to time some noise will occur on this line. Experience has shown that when this occurs there is a burst of errors usually lasting for at least a few milliseconds. Single errors are rare. The mean-time-between-error-bursts is the important criterion for the noise on the line, and the error recovery and detection algorithms have been designed with this in mind. Experience has also shown that this number will vary from line to line and from time to time on the same line. It is reasonable to assume that the mean-time-between-errors will be about 20 seconds for the 4800-baud line. (1 error burst every 10^5 bits). This should be considered a conservative estimate as we might well get error rates 1 or possibly 2 orders of magnitude better than this. It is also possible that error bursts may occur with greater frequency than average (especially at a high usage time). It is also possible that the line may break down completely and the error recovery system should allow a communication to continue at a reduced rate if this happens.

The Error Detection and Recovery Algorithms

There are two computers, A and B. A can send information to B over a 4800-baud (600 characters per second) telephone line AB. Similarly B can send information to A on a similar line BA. The line is synchronous, and if a character is not supplied to the line in time, the output multiplexor will automatically insert a SYN character. Thus the basic unit is the character. Another basic unit, called the block, is a sequence of T characters, the first D of them data, and the next C of them check characters (D = 13, C = 2, T = 15).

There is an input subroutine called MXIN at the input to the multiplexor line, and an output subroutine called MXOUT at the other (output) end of the communications line for line AB. Line BA has the same subroutines associated with it, but for now we will consider how transmission is done from A to B.

Let us look at Figure 1. The round boxes are subroutines and the square boxes are buffers. An arrow to a round box indicates flow of control, to a square box, flow of data. FRED is a 20-character input buffer to the Error-Free Communications Line (EFCL). It is loaded by FEEDFRED (which is not part of EFCL, but is called by MXIN).If FRED is empty, FEEDFRED is called to put something into FRED. Thus we can consider FRED as always having something in it. (FEEDFRED will be required to provide at least one character when called.)

The input subroutine (FEEDFRED) is allowed to output any characters with the exception of NULL. This identical character will appear in HARRY in the Remote Concentrator at a later time, and STEALFROMHARRY will be called to empty HARRY and process the string.

Let us now consider how normal transmission takes place. MXIN is called when JACK, the multiplexor hardware buffer, is empty. MXIN will then get a character from FRED and put it into JACK. When a character is taken from FRED the checksum in CHECK is recalculated. If the number of characters that have been taken from FRED since that current block began is D characters, a check character is output to JACK. If the number of the character that has now been output is now T, then the Number of Characters in The Block (NCIB) is reset to zero, and the checksum is zeroed. This ends the transmission of a block.

The checksum is two characters long, one being the exclusive OR of the first D characters of the block, and the second being computed the same way except that each time the checksum is recomputed, it is cycled right once. A third checksum character is contemplated--computed by a right cycle of 3. If the check character is calculated as zero, it is replaced with a 1 because a zero may not be sent except during resynchronization.

At the same time MXIN outputs a character from FRED or from CHECK, it will put the same character into LARRY, a 75-character ring buffer. Eventually the character will appear in JILL, and MXOUT will be called to remove the character from JILL and exchange it with the current character in JOE, a buffer containing T characters. The exchanged character is then put in HARRY.

Whenever a check character is put into JOE, the exchanged character is compared against its calculated value and no character is output to HARRY. The check character agrees in the error-free case.

Hopefully, the checksum will not always agree. We have detected an error and must get it corrected. We do it simply by sending a Message on line BA telling MXIN to stop reading from FRED and to retransmit the characters that it has been saving away in LARRY. When we analyze the algorithm we will see that LARRY will have saved characters from far enough back to be able to retransmit the block in error.

But the purpose of this section is to understand what the error recovery algorithm does, not why it works. Two things must now be defined, Messages and Acknowledgments. Both are started in the same way; the block currently being sent (if any) is aborted (stopped), and the Message or Acknowledgment to be sent is loaded into DAVE, preceded by three NULLs and a SYN character. A Message specifies the block in which the error was detected. An Acknowledgment, sent in response to a

Message, indicates the number of the block that follows it (the one that was requested by the Message). Immediately after the Acknowledgment is the block that was in error. Subsequent blocks follow in order.

There are two normal sequences of Messages when an error occurs, depending on whether the error occurred in one direction or both. They are shown in Diagram 1. Figure 2 shows the possible state transitions. A complete analysis is given in the next section.

<u>WARNING</u>: The preceding discussion has focused on the possible states of a single <u>line</u> and its associated buffers and send routines. What follows concentrates on a single <u>computer</u>, which has to deal with two lines: its input line, from which it is reading characters, and its output line, to which it is writing characters.

	Cause	Effects	State of A (before/after)	State of B (before/after)
CASE I:				
a)	A Detects Error in Block n	Aborts block m, Message (n)	B/C	B
b)	B Receives Message	Acknowledge (n)	C	B/M
c)	A Receives Acknowledgment	Acknowledge (m)	C/B	M
d)	B Receives Acknowledgment		B	M/B
CASE II:				
a)	A Detects Error in Block n,	A Aborts block p, Message (n)	B/C	B
b)	B Detects Error in Block m	B aborts block q, Message (m)	C	B/C
c)	B Receives Message	Acknowledge (n)	C	C/M
d)	A Receives Message	Acknowledge (m)	C/M	M
e)	A Receives Acknowledgment		M/B	M
f)	B Receives Acknowledgment		B	M/B

Possible states of A or B

B:	Both transmitting and receiving successfully (may include retransmission of earlier errors)
C:	Neither receiving or transmitting (waiting for a resynchronization)
M:	Transmitting successfully, not receiving (waiting for a resynchronization)
I:	Impossible - receiving but not transmitting

DIAGRAM 1

By analyzing this diagram, one should be able to understand what is going on in a simple case. But what happens if Messages or Acknowledgments are in error? They are ignored. Whenever a Message or an Acknowledgment is sent and the computer that sends it is not both receiving and transmitting, a timer is set to timeout one round trip later if a Message or Acknowledgment is not received. Thus if a failure occurs, MXIN will repeatedly send Messages until it gets an Acknowledgment.

Analysis of the Error Algorithm

We would like to define some constants for the communication line. In particular, the One Way Time (OWT) must be defined. The One Way Time is the amount of time it takes a character given to MXIN to be output by MXOUT, given that no error occurs. This time is determined by adding 30 milliseconds (3000 miles) plus 25 milliseconds (a 15-character Block Length). The result is 55 milliseconds, (35-character times). The round trip error time is just twice this: 70 characters or 110 milliseconds.

In order to understand what is happening we define the three states a computer can be in for error recovery: Bliss, Confusion, and Maine. In Bliss, the normal state, MXIN for the output line is outputting to JACK, and MXOUT for the input line is putting input in JOE. In the state of Confusion, MXOUT is looking for a resynchronization, and MXIN is not outputting anything. In the state of Maine, MXIN is outputting characters, but MXOUT is looking for resynchronization. The state where a computer is happily inputting but not outputting is not possible because in such a state it would never get a message telling it to begin transmission.

There are two obvious cases. (See Diagram 1). In the first an error occurs on the input to A. It goes into the state of Confusion after sending a Message to B asking B to start re-transmission with the block number found in error. This block is identified by the block number mod 128. When B gets the Message, it has transmitted a few blocks beyond the one that was in error. It sends an Acknowledgment to A indicating it is beginning retransmission with the requested block number. B, which was in the state of Bliss until it received this Message, goes into the state of Maine.

We left A in a state of Confusion. Shortly, it receives an Acknowledgment from B and resets the output block count to begin receiving the block that it had found in error (and those following it). The block number that it is given on input is redundant (it is the one that it expected to get anyway). A now sends an Acknowledgment to B and resumes the transmission starting with the block after the last block that it successfully transmitted, and goes into a state of Bliss. B, which was left in the state of Maine, soon gets this input and goes back to the state of Bliss.

The other simple case occurs when simultaneous errors occur on both lines. In this case both A and B change from a state of Bliss to one of Confusion, sending a Message indicating an error, sending an Acknowledgment, and beginning retransmission with the block indicated to be in error. Shortly thereafter each of these computers changes to the state of Bliss when it receives the Acknowledgment.

We should now understand the simple case when errors occur only in normal transmission. A Message indicating an error or an Acknowledgment not properly received is ignored by the receiving computer (B). The sending computer (A) has a one round trip timer. When the timer goes off, A retransmits exactly what it sent when it went into the state of Confusion initially. The timer is then reset.

If a line goes dead, both computers keep retransmitting Messages indicating the error each round trip time (1/8 of a second). When the line is restored, the Message will get through and retransmission will continue as if nothing had happened. Some input buffers might be full; but no characters will be lost by the EFCL.

Another possibility is that one computer could go into the state of Maine, sending characters to the other computer but not receiving any (successfully). In this case it is possible for the Acknowledgment that it sent when it went into this state to be lost. If this happens, the same timer (after 1/8 of a second) will cause the computer to retransmit the Acknowledgment and the blocks following it. If an error is detected by one computer, the other computer will either find out about it 1/16 of a second later (because it receives a Message) or the other computer will go into a state of Confusion (because it receives checksum errors).

In either case output will be stopped. Similarly, when in the state of Maine, if a computer does not receive an indication that the other computer is happily receiving blocks, it will time out and resend the Acknowledgment and the blocks in error. (The indication is successful reception from the other computer). So even if a line is cut off at this point, one (or both) computers would keep sending an Acknowledgment and the following blocks would be retransmitted every 1/8 of a second.

For this scheme we need an output buffer large enough to hold a round trip times worth of characters.

The number of characters of bandwidth lost due to an error is one round trip time plus the length of the error Message (or Acknowledgment). This overhead, which occurs in both directions for either Case I or Case II, can be verified by a straight forward analysis.

Assuming that the checksumming device shown has no subtle flaws, the chance of an undetected error is $1/255^2$ or about one undetected error for every 65,000 detected errors. Assuming the error will affect all of the teletypes, which seems reasonable, and the error rate is one burst each 10^5 bits (about every 20 seconds), one undetected error should occur every 10 days of continuous usage (for each Remote Concentrator), If the error rate is two order of magnitudes better as claimed by MILGO , then one error should occur every three years (actually every three months on some Remote Concentrator). If the error detection porbability is too low, an increase in the number of checksum characters to three should increase the mean-time-between-undetected errors to about once a year for all Remote Concentrators or once every twelve years for each Remote Concentrator. This assumes the figure of one burst error every 10^5 bits.

The preceding assumptions are conservative. Some errors can be detected before they do damage, and a great majority affect only one user. If only one user is affected, the mean-time-between-failure for a user is increased by a factor of about fifty.

The block size has no noticeable affect on the mean-time-between-undetected errors. Because errors come in bursts and the resulting block is garbage, then the checksum's correctness is $1/255^2$ as all transmissions are equally likely. Thus smallness offers no advantages. If there are two blocks rather than one affected by error, the chance that at least one will falsely be received correctly is $2/255^2$. But only the first block is of inport so we get the same result.

The Valparaiso Principle

We now add a governor to the communications line. Consider Figure 3; at computer A we put a counter called AC, and at computer B we put a counter called BC. AO is the block number being output, AI is the block number being input, AB is the error-free communications line from A to B. Thus,when AB is used in an equation it references the number of characters in the line AB. BA, BI, BC, and BO are similarly defined. AB includes the buffers as well as the physical line.

An initialization time

$AI = AO = BI = BO = BC = 0,\ AC = RTT,\ BA = AB = 0$

We are going to add a set of rules on the communications system:

1) If a character is input from BA, (put in HARRY)

$AI \leftarrow AI + 1,\ AC \leftarrow AC + 1,\ (BA \leftarrow BA - 1)$

2) A character may be taken from FRED only if AC >0;
in which case:

$AO \leftarrow AO + 1,\ AC \leftarrow AC - 1, (AB \leftarrow AB + 1)$

The expression in the parenthesis is not performed by the computer but represents the data entering or leaving the line. The same rules are followed by computer B.

The effect of these rules is to prevent transmission from proceeding in one direction faster than in the other direction. The implementation is straight forward:

AC (and BC) must be kept updated, and characters may not output if AC (or BC) is zero. In this case (AC=0) the error-free communications line inserts a SYN character (which is not counted in the block length, not put in LARRY (although it could be), and is completely ignored by the receiving counter).

We would like to fix a maximum time for a message sent from A, to be received by B, and for the reply to the message to be received back at A. There is such a time, called VRTT where VRTT = RTT + m + r. M is the maximum length of message, and r the maximum length of the reply. This is called the Valparaiso principle and is now stated:

> Let the block number being input to computer A be N, and the Valparaiso round trip time (defined above) be VRTT. If A sends a message to B, then:
>
> 1) When block N plus VRTT begins to be output by B, B will have gotten the message,
> 2) When block N plus VRTT begins to arrive at A, A can be assured that B has gotten the message,
> 3) When block N plus VRTT begins to arrive at A, any reply that B might send to A has been received by A (assuming B replies immediately). This implies that

when the last character of A's message emerges from the EFCL in B, there is still room for all the characters of B's reply in blocks earlier than block N plus VRTT.

4) If no reply is received in this time, there will be no reply.

In order to show that this is true, let us consider the following equations which are valid initially:

(a) $AC + AB + BC + BA = RTT$

(b) $BO = AI + BA$,

(c) $AO = BI + AB$,

(d) $BO = BI + BC$,

(e) $AO = AI + AC$,

(f) $\emptyset \leq AC \leq RTT$

(g) $\emptyset \leq BC \leq RTT$

(h) $\emptyset \leq BA \leq RTT$

(i) $\emptyset \leq AB \leq RTT$

We now observe that equations (a) to (i) are valid for all time (if they are valid initially) because the transformations that are allowed (1 and 2) preserve their validity. With this in mind, we will now derive BO_3, the number of the block being output by B after it replys to a message sent when $AI = AI_1$.

First we note:

$BC_2 - BC_1 = (BI_2 - BI_1) - (BO_2 - BO_1)$

Rearranging terms, we have:

$BO_2 = BO_1 + BC_1 - BC_2 + BI_2 - BI_1$

But,

$BI_2 \leqslant BI_1 + AB_1 + m$

where AB is the number of characters in the line AB when the message was inserted, m is the maximum length of the message and BI_2 is the value of BI when the message is received at B. Now we have:

$BO_2 < BO_1 + BC_1 + AB_1 + m.$

We have deleted BC_2 from the equation because of equation (g). A's reply is sent immediately, being finished at time $BO_3 = BO_2 + r$, where r is the maximum length of the reply.

Thus we can now say that:

$$BO_3 < BO_1 + BC_1 + AB_1 + m + r.$$

but, by equation (b)

$$BO_1 < AI_1 + BA_1$$

So,

$$BO_3 < AI_1 + BC_1 + AB_1 + BA_1 + m + r.$$

Because of equation (a) we can say:

$$BO_3 < AI_1 + RTT + m + r - CA_1$$

because of (f)

$$BO_3 < AI_1 + VRTT,$$

where VRTT is defined by:

$$VRTT = RTT + m + r;$$

We can also note that:

$$AI_4 = BO_3 < AI_1 + VRTT$$

where AI_4 is the time that A has finished receiving the reply.

This last statement and the definition of VRTT just before it is the mathematical formulation of the Valparaiso principle.

The longest message reply is 3 characters. Both m and r must have two check characters included as part of their maximum length. This proof has assumed that all replies can be sent immediately. In fact, this is not always possible because the last character sent might be SHIFT1. This delay (worst case 2 characters) is incorporated into r, making:
RTT = 5 blocks, m = 5 characters, r = 7 characters. Because AC and CB are kept as block counters rather than character counters, 1 block must be added to this, making:

VRTT = 7 blocks.

Let us consider a simple example assuming the Valparaiso round trip time is therefore 7 blocks. If the current block being input to computer A is, say 9, then if a message is sent to B, the block being assembled for output in B when the message is received will be block 16 or less, and then will be enough room in block 16 for the reply. When block 17 is being assembled in B, the message will have been received by B. When block 17 is being received by A, any reply that was sent by B in reply will have been received. Thus we see the four aspects of the Valparaiso Observation:

1) When block 17 is being assembled by B, B has received the message.
2) When block 17 begins to arrive at A, A knows that B has received the message.
3) When block 17 begins to arrive at A, A has received its reply to the message

(if any); i.e. the reply was in block 16 or earlier.

4) If A has gotten no reply, then when block 17 begins to arrive, A knows B sent no reply.

EFCL Usage

We have described the method of error detection and recovery. We now have, in a certain sense an error free-communications line. Let us try and get a feeling for what we have and how we can use it.

There is a line from A to B, and another from B to A. Associated with each of these lines is an input counter (clock) and an output counter. We can look at this line in the following way. Each 1/600 of a second each input counter increases by one and each output counter does the same. These counters are not synchronized with each other, but an input counter tends to differ from the output counter for the same line by about OWT characters. The counters on the line AB may drift relative to the line BA counters. If we remember this, we may use these counters or the corresponding block counters.

The communications line clock may stop for several cycles from time to time. In particular it is very likely to stop for one round trip time (55 cycles) or more when an error occurs. Another part of the system, such as the output line scanner, can use the communications line counters (clock), but it must realize that the counters for the EFCL may come to a stop while the counter (clock) for another system continues as before. This will make an analysis of some algorithms in the communications system easier.

And If A Line Drops Out

The preceding scheme works fine if there is an acceptable line to transmit data on. If a line breaks down, there must be some backup. This is provided by connecting two remote concentrators together with a 2400-baud line. If the line to one of the remote concentrators goes down, then the line to other remote concentrator is used to send data to both remote concentrators (with the help of the 2400-baud line).

This is known as the triangle scheme. This scheme was adopted because the minimum amount of hardware for a much more reasonable method is not available. Triangles might make for good drama, but they don't make for elegant system design.

Whenever an error occurs, and that includes the errors initiated by line timeouts, a CPU process is awakened and given the appropriate information. This process might decide that a line is completely useless (or so close to useless that it should be considered as useless). When this happens, this process rearranges the remote concentrator tables in the CHIO core to put the two concentrators in triangle mode.

In this mode (called half-fast mode) characters can enter MXIN (in the CHIO) from FRED (as before) or from LISA, the name that I have chosen for the buffer LARRY for concentrator C. MXIN inserts ORC as the last character before the checksum character in a block the control character if the next block is destined for other remote concentrator. MXIN for concentrator C will fill up LISA from FRAN (which corresponds to FRED) and will not output characters on the 4800-baud line AC. MXIN for the working 4800-baud line AB will make up a block out of characters from FRED, inserting the ORC character in it. It will then switch and take the next block out of LISA, alternating between the two blocks.

The advantage of this method is that it allows a convenient method not only of alternating blocks, but of turning on and off the half-fast mode. Half-fast mode is turned on by setting switches in each remote concentrator table. The switch for communications line AB causes the CHIO to take blocks alternately from each computer. The switch for line AC prevents it from outputting characters on the line. As soon as LISA is emptied by this process, characters are then input from FRAN, the buffer corresponding to FRED. Communications on line AC can be resumed by waiting for the Valparaiso round trip time to occur on line AB without multiplexing any information from AC on it. Output can then be resumed on AC starting with an empty LISA.

Meanwhile at the other end, MXOUT has two switches, OTC1 and OTC2. Normally both are off, but if an ORC is received by the Remote Concentrator, OTC2 is set. After a block has successfully been removed from JOE, OTC1 is set to OTC2 which is cleared. If OTC1 is set, characters (including check characters) are taken out of JOE and put in PIERRE, a buffer RTT characters long. At the end of this block OTC1 is reset.

There is another task in the remote concentrator that is driven by input from, or output to, the 2400-baud line connecting the two computers. If the Remote Concentrator pair is not in half-fast, mode it does nothing but send useless messages back and forth, sending an error message to the CHIO whenever an error is detected. When the triangle scheme is invoked, this subroutine, called ANDRE will take characters out of PIERRE and put them on the 2400-baud line in the same way that MXIN would take characters from LARRY to put on the 4800-baud line.

If an error occurs, a message is sent to the CHIO asking it to stop transmission to the 2400-baud line and the previously described error procedure is followed for the 2400-baud line. When PIERRE is emptied, a message is sent to the CHIO telling it to resume sending as before.

If the 2400-baud line goes down, then PIERRE will have RTT characters in it that have to be retransmitted, where RTT is the round trip time of the 2400 baud line. PIERRE will continue to be filled with characters until the CHIO stops sending them, which will be VRTT characters later because of the Valparaiso observation. At most, only half of these characters go onto the 2400-baud line so the total size that PIERRE must be is VRTT/2 + vrtt or about 160 characters.

When the buffer PIERRE has been emptied, a character is sent to the CHIO to start retransmission to the 2400-baud line. This means that the efficiency of the system is not as good as it could be because the 2400-baud line is occasionally waiting for input. The bandwidth of the 4800-baud line has not been wasted, however, because it transmits at full speed to one remote concentrator if the 2400 baud-line is in error mode. Remote Concentrator C gets these characters and puts them in JOSE. MXOUT takes these characters and treats them as if they had come from JILL.

Input from computer C in half-fast mode is done in the same way. A look at Figure 4 will show how the buffers and sub-routines are arranged. HARRIET is an input buffer for line AC comparable to HARRY. MXOUT delivers the character to the correct buffer depending on the input switch setting. At the input end ANDRE gets characters from AUGUSTE, the HARDWARE BUFFER on the 2400-baud line and puts them in PIERRE. MXIN, in half-fast mode takes blocks of characters alternately from FRED and PIERRE. If a block comes from FRED, the last character in the block is the control character ORC.

The buffer PIERRE must be the same length as the output PIERRE. The same analysis will show this. Similarly, after a resynchronization, PIERRE is emptied before MXIN goes into half-fast mode.

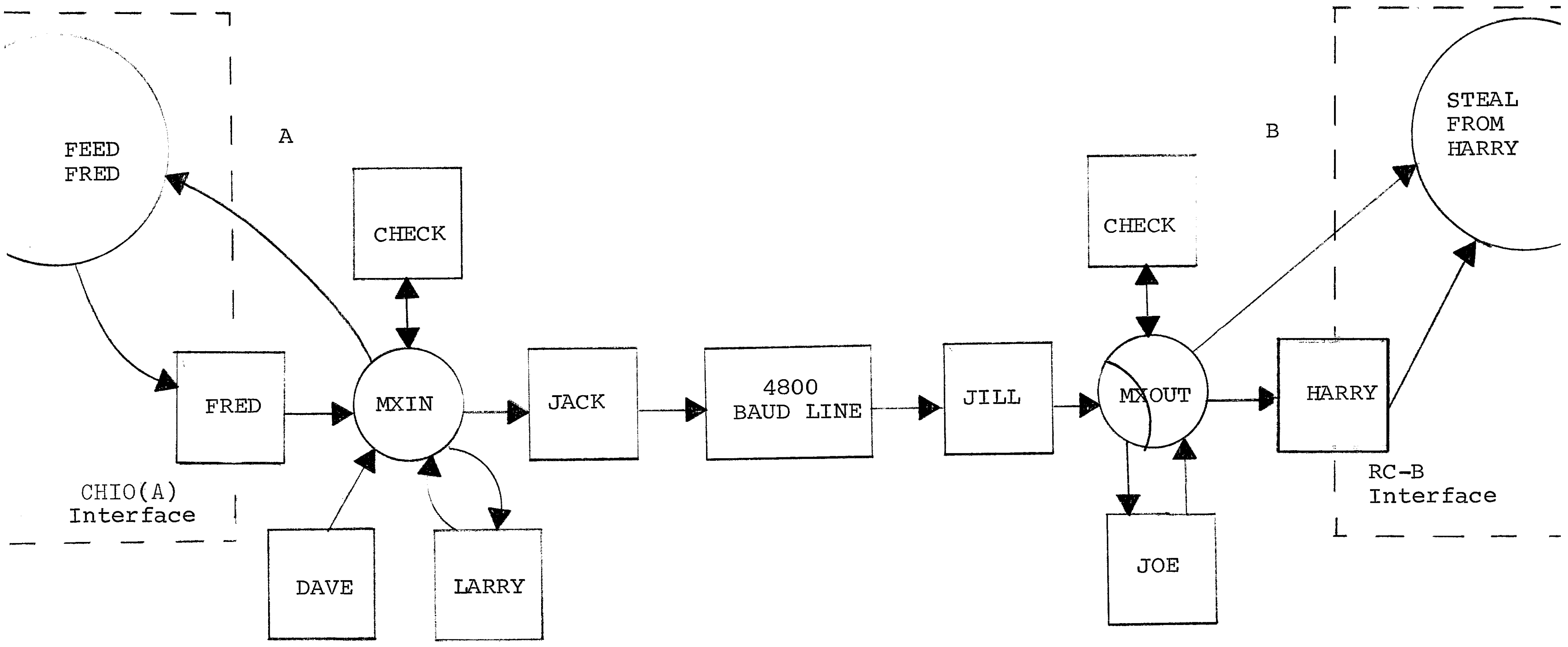

Figure 1
Error-Free Communications Line Flow A to B

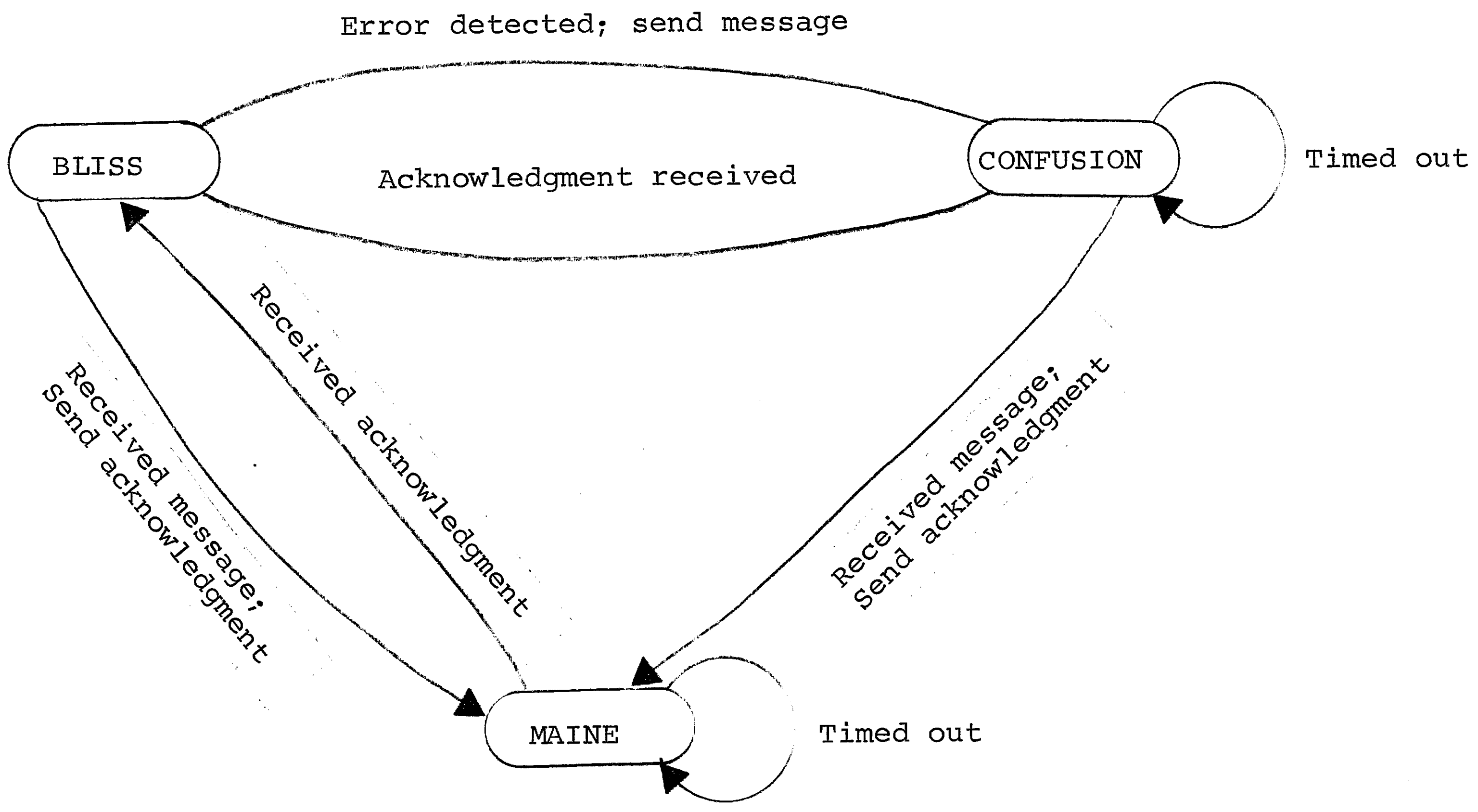

Figure 2
State Diagram for Error Subroutines

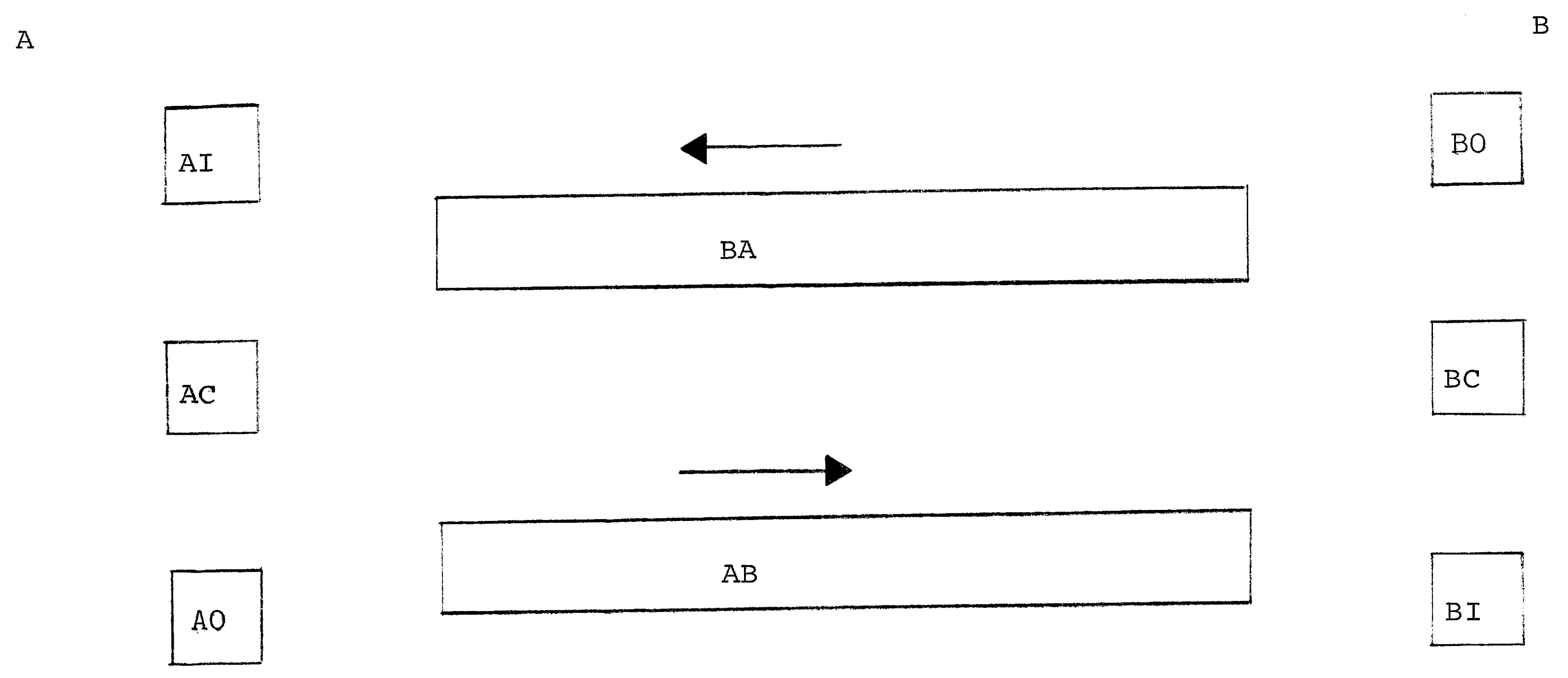

Figure 3

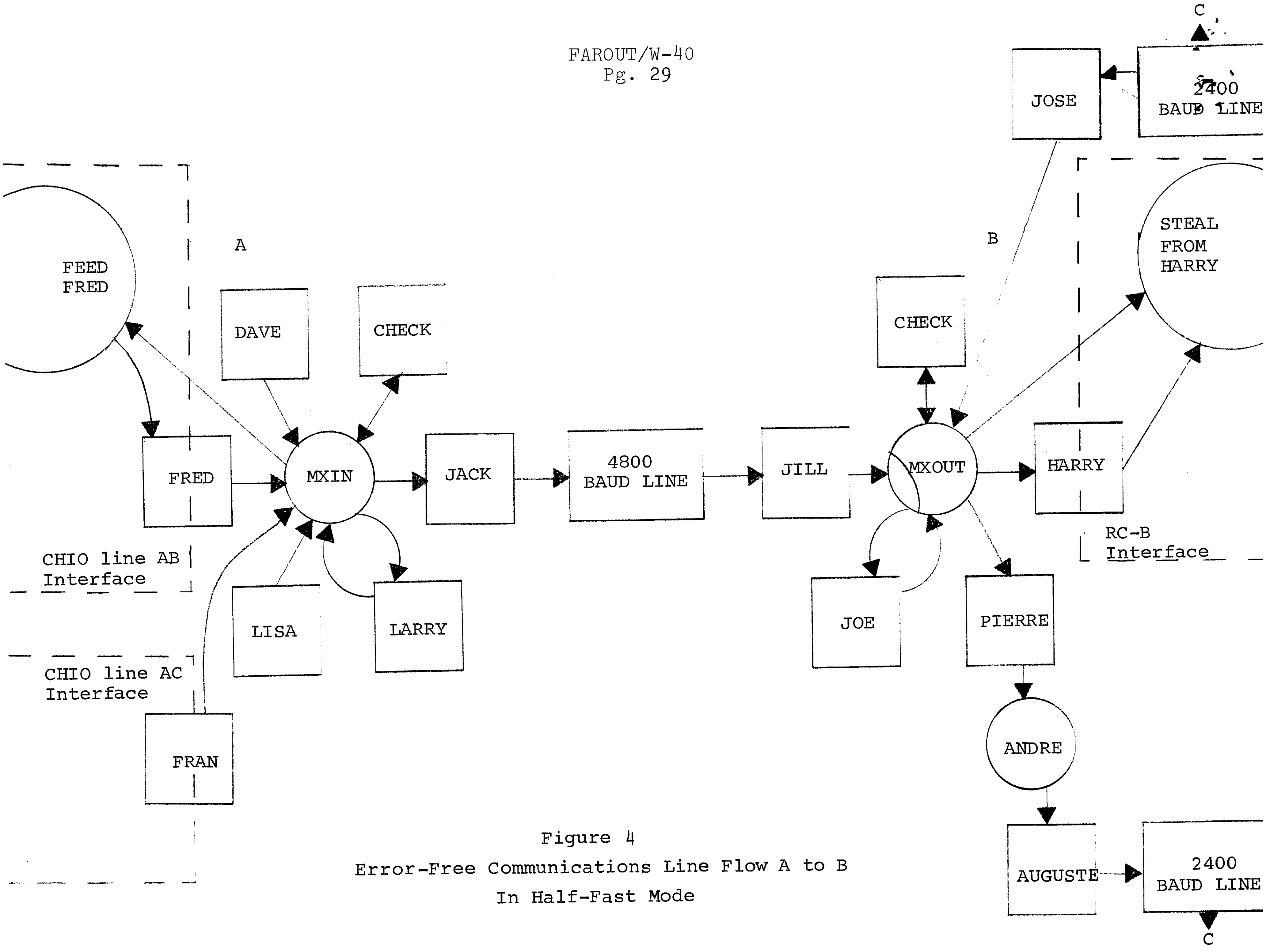

Figure 4
Error-Free Communications Line Flow A to B
In Half-Fast Mode

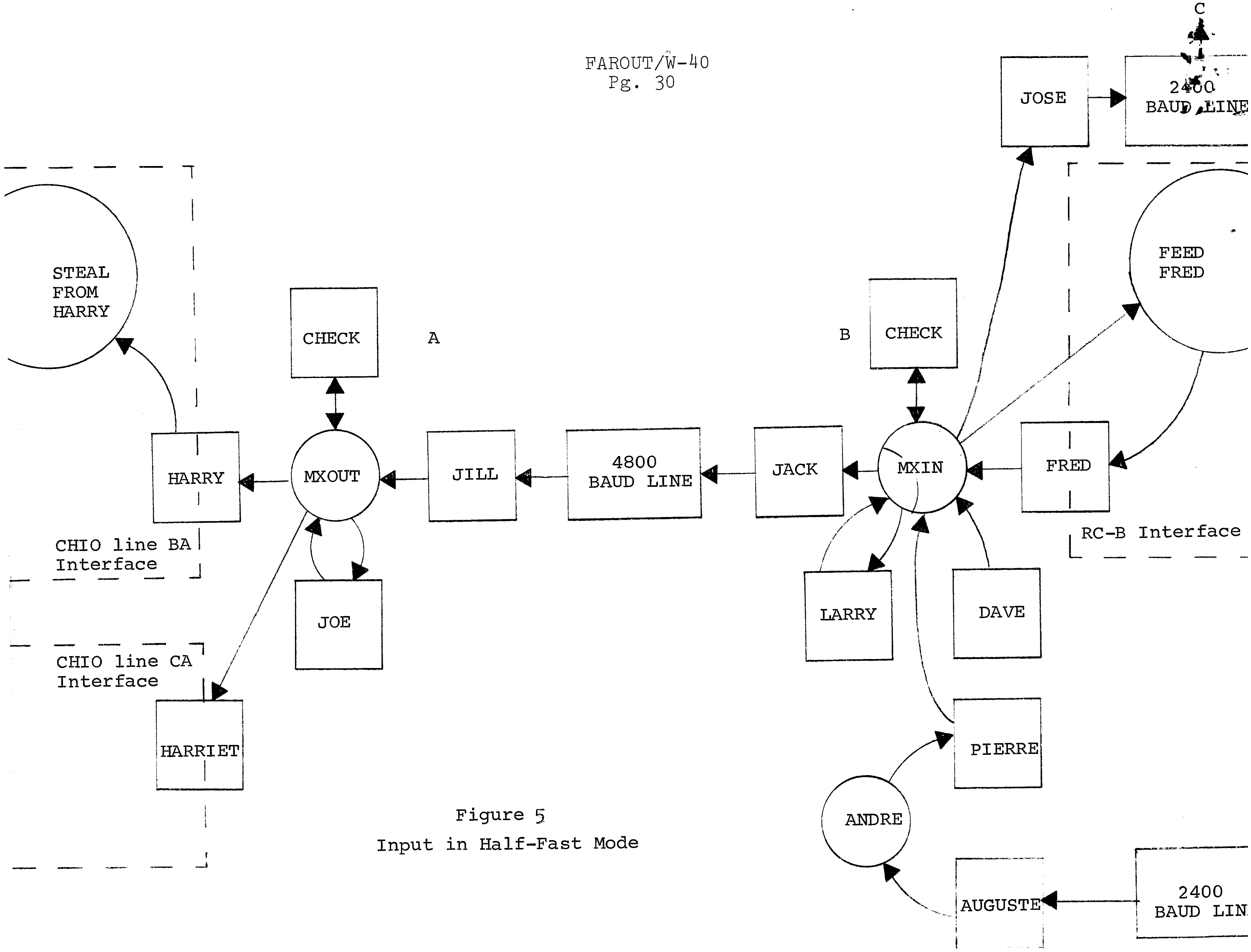

Figure 5
Input in Half-Fast Mode