DELIVERY AND DISCRIMINATION: THE SEINE PROTOCOL

M. G. Gouda
University of Texas
Austin, Texas 78712

N. F. Maxemchuk
U. Mukherji
K. Sabnani
AT&T Bell Laboratories
Murray Hill, NJ 07974

ABSTRACT

We present two protocols for information exchange between multiple identical senders and a single receiver. At each instant, every sender sends one bit, and the bits from all of senders are or-ed together into one bit before being received by the receiver. If a sender has a data message to send, it sends the message bits one by one; otherwise it sends zero bits. Clearly, if the sending of two messages by two senders overlap, then the resulting "collision" can result in a corrupted message, i.e., one that was not sent by either sender. The function of the protocol is to deliver those and only those messages that are not corrupted by collision. (In other words, the receiver acts as a discriminating seine that catches and delivers only uncorrupted messages; hence the title.) The two protocols presented here are based on Manchester codes and general balanced codes, respectively.

1. Introduction

In this paper, we present two protocols that can be used for information exchange in some fiber-optic networks [Ma87a, Ma87b, MMWS88]. In such systems, the bit streams from multiple senders[1] are or-ed together on a bit-wise basis, into one stream. This stream is then delivered to every receiver in the system. The function of these protocols is to allow each receiver to recognize and extract from its incoming bit stream those and only those data messages that are not corrupted by collision with concurrently sent messages.

These protocols, called the *Seine protocol* and the *Seine protocol generalized*, provide a delicate balance between delivery and discrimination. On one hand, these protocols allow all those messages that are not corrupted by collision to be accepted and delivered. On the other hand, these protocols are discriminate enough not to let any of those that are corrupted to be accepted. The

1. Although there are different propagation delays from each sender to each receiver, these delays are not important as shown in Section 7. The major reason as explained later is that the activities of all senders are independent of one another.

Seine protocol can be used in some fiber-optic networks [Ma87a, Ma87b, MMWS88] to choose message format and designs of the receiver and the transmitter.

The rest of this paper is organized as follows. In Section 2, we formally define our problem. In Section 3, we give an informal description of our solutions. Our first solution is the Seine protocol based on the Manchester coding procedure [MB76]. A formal definition of the protocol and a proof of its correctness are given in Sections 4 and 5, respectively. A generalization of the Seine protocol followed by a sketch ot its correctness proof are given in Section 6. This generalization is a solution based on the general class of balanced codes [BOS86, Mo83, GS80]. Some applications of the Seine protocol are outlined in Section 7. We end with some concluding remarks in Section 8.

2. *The Problem*

It is required to design a communication protocol between m identical *senders*, $m \geq 1$, and one *receiver* (Figure 1).[2] At every instant, each sender sends one bit and the receiver receives one bit. The value of the bit received by the receiver at any instant equals

$$b_0 + b_1 + \cdots + b_{m-1}$$

where b_i is the value of the bit sent by the ith sender at that same instant, and "+" is the boolean "inclusive or" operator.

Most of the time a sender has no string of data bits to send, in which case it continues to send zero bits. At arbitrary instants, a sender gets an arbitrary, but finite, string x of data bits to send. At such an instant, the sender first encodes[3] the string x in some appropriate form called the *message of* x, then sends the bits of the message, one by one. At the end, the sender returns to sending zero bits, until it gets the next string of data bits to send, and the cycle repeats. Henceforth, the message of x is denoted msg[x].

The receiver receives the bits of the incoming stream one at a time. When the receiver recognizes a contiguous block of received bits as msg[x] for some bit string x, it accepts x.

The protocol between the senders and the receiver is required to satisfy the following two conditions:

1. *Delivery*: If a sender sends the bits of some message msg[x] while the other senders send zero bits, then the receiver will accept x.
2. *Discrimination*: If the receiver accepts a string x, then one of the senders must have sent msg[x].

A design of the required protocol should define: the encoding of sent messages (i.e., define msg[x] for any bit string x), and the designs of the sender and the receiver. But before we present our design of the protocol in Section 4, we discuss in the next section ideas that led us to that design.

3. *Search for a Solution*

Each message sent by a sender (msg[x]) has the structure: preamble code[x] postamble where the preamble is a fixed bit pattern which must constitute the start of each message, code[x] is an encoding of the string

2. We will show in Section 7 that the first protocol presented here can be used for a fiber-optic network [Ma87a, Ma87b, MMWS88].

3. As described earlier, we will consider two encoding procedures: the Manchester coding procedure and a general balanced coding procedure.

x using an appropriate coding procedure, and the postamble is a fixed bit pattern which must constitute the end of each message. Solving this problem involves choosing an appropriate value for the preamble and the postamble, and designing a receiver. We will consider two coding procedures: one based on the Manchester coding procedure and another based on a general balanced coding procedure. Manchester coding procedure is very widely used because of its self-clocking property and its simplicity. For example, Ethernet [MB76] uses this coding procedure. The solution for this code is called the Seine protocol and is described in Section 4. Although the Manchester code is a special case of a balanced code (in which each code word has equal number of zeroes and ones), it is not very efficient. Some other balanced codes are more efficient than Manchester codes as shown in Section 6. In Section 6, we give a solution for the general balanced coding procedure, called the *Seine protocol generalized*. Although the Seine protocol is a special case of the Seine protocol generalized, we describe both of these protocols. We also present the correctness proof for the Seine protocol. The proofs for the Seine protocol generalized given in [GMMS88] are similar to those for the Seine protocol.

The values of preamble and postamble were chosen by trial. From this exercise, we came up with the following generic structure for these bit strings. The bit string preamble has the structure: (string of 1's) (string of 0's) 1. Let us denote the length of the longest string of consecutive 1's in the bit string 1code[x] as i. The length of the leading string of 1's in the preamble must be larger than i. Thus, the string of 1's in the preamble has at least i+1 bits. The string of zeroes is longer than the longest string of zeroes in code[x]. Let the length of the longest string of 0's in code[x] be denoted as j. Then, the string of 0's in the preamble has j+1 bits. This string allows the receiver to determine when two senders start messages one bit apart. The trailing 1 in the preamble ensures that the corrupted message will have only j bits and will be discarded by the receiver.

The postamble marks the end of a valid message. It has the structure: (string of 0's). This string of zeroes is longer than the largest string of consecutive zeroes at the beginning of a code word. Both of the coding procedures used here belong to the class of block codes. Each word of r bits in x is coded into a *code word* with l bits.

The receiver designs given later in the paper are self-synchronizing [Ko78]. Reception of the preamble starting in any state drives this machine into a known state. Starting from this known state, the receiver can decode a valid code[x] and compute x.

4. *The Seine Protocol*

We start by defining msg[x] for any bit string x. We then use this definition to design the sender and receiver for this protocol.

4.1 *Messages.*

Let $x = x_0 x_1 \cdots x_{n-1}$ be a string of n bits. We define msg[x] to be a string of 2n + 9 bits of the form:

$$\text{msg}[x] = 1110001\ y_0 y_1 \cdots y_{2n-1}\ 00$$

where for $i = 0, \ldots, n-1$, $y_{2i} = \neg x_i$ and $y_{2i+1} = x_i$, and $\neg$ is the usual "invert" operator. Thus each bit x_i in x is represented by two bits y_{2i} and y_{2i+1} in the string $y = y_0 y_1 \cdots y_{2n-1}$. Bit y_{2i} is the inverse of x_i and bit y_{2i+1} is the same as x_i. String y is usually referred to as the *Manchester code* of string x. Henceforth, we refer to string y as code [x] and to string 01y as code [1x]; in particular,

$$\text{msg}[x] = 1110001\ \text{code}\ [x]\ 00.$$

Notice that code [x] has 2n bits iff x has n bits. Moreover code [x] is *balanced*, i.e. it

has equal numbers of zeroes and ones, irrespective of the value of x. In msg[x], the preamble is 1110001 and the postamble is 00.

4.2 Sender.

As specified by the problem, when the sender has no string of data bits to send, it sends zero bits. On the other hand, when the sender has a string x of data bits to send, it sends msg[x] one bit at a time (starting with the three ones at the beginning of msg[x]).

4.3 Receiver.

The receiver is defined by the state-transition diagram in Figure 2. This model falls in the category of extended finite state machine models [BS83]. In this figure, each state of the receiver is represented by a small circle, and each transition is represented by a directed labeled edge. An edge label is either <bit value> or <bit value>/<action>, where a label <bit value> can be viewed as a special case of the label <bit value>/<action> when <action> is the null action *skip*.

The semantics of a transition is as follows. A transition labeled v/c from state s_i to state s_j means that if the receiver is at state s_i and receives a bit with value v, then action c is executed and the receiver moves to state s_j. The execution of an action consists of updating or ''accepting'' the current value of a local variable rcvd of type bit string. This local variable rcvd stores the intermediate value of a received message. Its value is "accepted" if the incoming message is valid. For example, referring to Figure 2, executing the action rcvd := < > assigns the empty string to rcvd. Executing rcvd:=rcvd.0 concatenates one zero bit to the tail of rcvd. While executing accept x, the receiver "accepts" the string x.

The machine shown in Figure 2 has five states: s_0, s_1, s_2, s_3, and s_4. An interesting characteristics of the receiver machine is its self-synchronizing property. Starting from any state, reception of the bit pattern 11100 drives it into state s_2. This means that this machine can accept a valid message starting from any state. The portion of the machine which decodes the incoming encoded message also is used to recognize the last two bits 01 of the preamble.

5. Correctness of the Seine Protocol

Correctness of the Seine protocol is established by showing that the protocol satisfies the two properties of delivery and discrimination stated in Section 2.

Theorem 1: *(Delivery)* If a sender sends the bits of some message msg[x] while the other senders send zero bits, then the receiver will accept x.

Proof: Assume that a sender sends the bits of msg[x] for some string x while other senders send zero bits. Then the receiver will receive the bits of msg[x] unaltered. Since the bits of msg[x] constitute a string in the form 11100 code [1x] 00, it is sufficient to show that if the receiver is at any state (of its five states), and receives a string 11100 code [1x] 00, then it must accept x. The proof is divided into three parts, each one of them can be checked by referring to Figure 2. First, if the receiver is at any state and receives a string 11100, it must reach state s_2 with rcvd = < >. Second, if the receiver is at s_2 with rcvd = < > and receives a string code [1x], it must reach s_2 with rcvd = 1x. Finally, if the receiver is at s_2 with rcvd = 1x and receives a string 00, it must reach s_0 after accepting x.

Theorem 2: *(Discrimination)* If the receiver accepts a string x, then one of the senders must have sent msg[x].

Proof: We divide the proof into the following two parts.

a. If the receiver accepts a string x, then it must have received a string 00 code [1x] 00.

b. If the receiver receives a string 00 code [1x] 00, then one of the senders must have sent msg[x].

Part a: Assume that the receiver accepts a string x. Then it must have reached state s_0 with rcvd = 1x, or equivalently, it must have received a string 00 after reaching state s_2 with rcvd = 1x. This means that the receiver has received a string code [1x] 00 after reaching state s_2 with rcvd = < >. This in turn means that the receiver has received a string 00 code [1x] 00 (after reaching s_0).

Part b: Assume that the receiver receives a string 00 code [1x] 00. Since the first three bits in this string are 0's, each sender must have sent three 0's in these positions. (Recall that the bits sent in the same position are or-ed into one bit before delivery to the receiver.) Similarly, since the last two bits are 0's, each sender must have sent two 0's in these positions.

Now what could a sender have sent in the remaining positions? Clearly, no sender could have started a new message in these positions since this would require sending 111 which would have to be received unaltered by the receiver, and this violates the fact that the receiver has received in these positions code [1x] which cannot have the string 111. Therefore, in these positions, each sender must have sent either a string $00 \cdots 0$ or a string code [1y] $00 \cdots 0$, for some string y.

Without loss of generality, assume that code[1x] has 2n bits for some $n \geq 1$. Now if code [1y] sent by a sender in these positions has less than 2n bits by each sender, then the bits sent in the last two positions are 00, contradicting the fact that the last two bits in code [1x] are either 01 or 10. Therefore, at least one of the senders must have sent in these positions some code [1z] that has 2n bits.

It remains now to show that this code code[1z] is identical to code[1x]. This proof is by contradiction. Assume that this code [1z] is different from code [1x]. This means that code [1z] while being sent has collided with at least one concurrently sent code [1w], and the result was code [1x] different from code [1z]. However, collision can only increase the number of ones in the received bits. Since code [1z] is balanced, i.e. has equal numbers of zeroes and ones, then code [1x] must be unbalanced which is a contradiction. Therefore, code [1z] must be identical to code [1x].

So far, we have established that at least one sender must have sent a string 00 code [1x] 00. Then, this sender must have sent 111 previously, i.e. it must have sent msg[x].

6. *The Seine Protocol Generalized*

In the Seine protocol, each message msg[x] has 2n+9 bits, where n is the number of bits in the data string x. The dominant term "2n" is due to the fact that x is encoded using the Manchester coding procedure. To reduce this term, the data strings could be encoded in some other balanced code (i.e. one where each code word has an equal number of zeroes and ones) that is more efficient than the Manchester code (i.e. one in which each data string on n bits can be encoded in less than 2n bits). Note that the Manchester code is a special instance of balanced codes.

Next, we describe a general balanced coding procedure [BOS86, GS80]. For such a code, every data string, to be sent by the protocol must have a multiple of r bits, for some r>1. Each word of r bits is coded as a code word consisting of l bits with the ratio l/r less than 2. Such a balanced code is more efficient than the Manchester code. For example, if r=4, then the following balanced code with l=6 can be used to encode the data strings to be sent. It is based on an example given in [BOS86].

Data strings with 4 bits each	Corresponding balanced code words with 6 bits each
0000	001011
0001	001101
0010	010011
0011	010101
0100	010110
0101	011001
0110	011010
0111	011100
1000	100011
1001	100101
1010	100110
1011	101001
1100	101010
1101	101100
1110	110010
1111	110100

This code can be used for encoding any data string x into code[x] as described below. First, partition x into words of four bits each, then replace each word with the corresponding code word from the above table. For example, if x=00001111, then code[x] = 001011110100. Thus, each code[x] has 3n/2 bits, where n is the number of bits in string x. This represents a 50% improvement over a Machester code, where code[x] has 2n bits.

In the remainder of this section, we present a generalization of the Seine protocol for the case in which the data strings are encoded using a balanced code, not necessarily the Manchester code. Henceforth, let C be the set of code words that are used in the encoding.

6.1 Messages

From the given set C of code words, define the following three quantities:

i = the largest number of successive 1's in 1 code[x]. This number is the maximum of two numbers: number of successive ones in two adjacent code words in C, number of successive ones at the beginning of any code word plus one.

j = the maximum number of successive zeroes in 1 code[x]. This number is equal to the maximum number of successive zeroes in two adjacent codewords in C.

k = the maximum number of successive zeroes at the beginning of a code word in C.

(For instance, i=4, j=4, and k=2 for the set of code words in the above table.) These three quantities are used in defining msg[x] as follows.

msg[x] = $(1)^{i+1}$ $(0)^{j+1}$ 1 code[x] $(0)^{k+1}$
where code[x] is the encoding of string x using the code words in set C, $(1)^{i+1}$ is a string of (i+1) ones, and so on.

6.2 Sender

As before, when the sender has a string x of data bits to send, it sends msg[x] one bit at a time starting with the $(1)^{i+1}$ bits at the beginning of msg[x]. Otherwise, it sends zero bits. In msg[x], the preamble is $(1)^{i+1}(0)^{j+1}1$, and the postamble is $(0)^{k+1}$.

6.3 Receiver

The state transition diagram for the receiver is shown in Figure 3, where as before small circles or nodes represent states, directed edges represent transitions, and the edge labels define the actions that accompany the transitions. This model falls in the category of extended finite state machine models [BS83]. For example, the outgoing edge from from state s_{j+2} is labeled b/y:=y.b which means that the receiver waits at state s_{j+2} until it receives bit b then concatenates b to the end of the string y. Two local variables *rcvd* and *y* are used in this machine. The variable *rcvd* stores the intermediate value of the decoded incoming message. Its value is "accepted" if the incoming message is valid. The variable *y* stores the received prefix of the next incoming code word. If this incoming code word is valid, the corresponding decoded

word is concatenated to *rcvd*. This machine has j+3 states: s_0, s_1, $\cdots$, s_{j+2}. The reader should note that the FSM for the Seine protocol shown in Figure 2 is not a special case of the FSM for the Seine protocol generalized shown in Figure 3. The last two bits 01 of the preamble constitute a valid codeword for the Manchester code. The part of receiver which can recognize the last two bits of the preamble can be absorbed into the portion of machine which decodes code[x] for the Seine protocol. This reduction was done in the FSM shown in Figure 2.

The diagram has a diamond-shaped node labeled "y=?". This node represents a check by the receiver on the current value of y. Here is a list of the receiver's actions corresponding to four possible values of y:

a. *y = proper prefix of some code word in C*: In this case, the receiver moves to state s_{j+2}.

b. *y = some code word in C:* In this case, the receiver updates rcvd with the word that corresponds to the code word y, then makes y an empty string, and moves to state s_{j+2}. The decoded word corresponding to the code word y is represented as word(y). In other words, code[word(y)] = y.

c. $y = (0)^{k+1}$: In this case, the receiver accepts rcvd and moves to state s_0.

d. Otherwise, the receiver moves to state s_0.

The next two theorems establish that the Seine protocol generalized satisfies the delivery and discrimination properties stated in Section 2.

Theorem 3. *(Delivery of the General Protocol)* If a sender sends the bits of some msg[x] while the other senders send zero bits, then the receiver will accept x.

Proof: Given in [GMMS88].

Theorem 4. *(Discrimination of the General Protocol)* If the receiver accepts x, then one of the senders must have sent msg[x].

Proof: Given in [GMMS88].

7. Application to a Fiber-optic Network

The problem solved can used to design the message formats and receivers for a local area network shown in Figure 4, called the *D-network* [TC83, FBT81, Ma87a, Ma87b, MMWS88]. All the stations are connected by a uni-directional fiber link. Each station has a sender and a receiver. Multiple access protocols given in [Ma87a, Ma87b, MMWS88] are used to share this link among various stations. These protocols described in [Ma87a, Ma87b, MMWS88] permit collisions to occur. On the other hand, the protocols described in [TC83, FBT81] do not permit collisions. Transmission from each sender is or-ed onto the bit stream traveling on the fiber link. Once the transmissions generated by all senders have been or-ed together, the resulting bit-stream is received by all receivers. All receivers get the same incoming bit stream. We want to design a procedure which ensures the two properties of delivery and discrimination presented earlier.

Delivery: If a sender sends a valid message and all other senders send zeroes, then each receiver must accept this message. Suppose station *y* wishes to send a message to station *z*. It sends the data message with the address of station *z*. Any messages accepted by the receiver are checked if they are destined for the local station. If they are, then they are given to the local user or the next higher level of protocol. Otherwise, they are discarded.

Discrimination: If all receivers accept one message, one of the senders must have sent it.

The D-Network shown in Figure 4 can be modeled as a system shown in Figure 5.

Although there are different propagation delays from each sender to any receiver, they are not important. It is because the actions of all senders are not coordinated. Since the activities of all the senders are independent, all the delays before the Or-box shown in Figure 5 can be removed. Therefore, the system shown in Figure 5 is similar to the one shown in Figure 1. The solutions presented earlier apply to this problem.

8. *Concluding Remarks*

We have presented here an interesting problem for information transfer from many senders to one receiver, and two solutions. One of these solutions can be used for the design of *D-Network*, a fiber-optic network.

We assume here that all senders in the system are synchronized in their bit transmissions. This assumption is valid in some implementations of *D-Network* but is not valid for other implementations. In the future, it may be of interest to solve this problem for the asynchronous case.

REFERENCES

[BOS86] E. E. Bergmann, A. M. Odlyzko, and S. H. Sangani, "Half-Weight Block Codes for Optical Communications," AT&T Tech. Jour., vol. 65, no. 3, pp. 85-90, 86.

[BS83] G. V. Bochmann and C. A. Sunshine, "A Survey of Formal Methods," *Computer Networks and Protocols,* P. E. Green (Ed.), May 1983, pp. 561-578, Plenum Press.

[FBT81] L. Fratta, F. Borgonovo, F. Tobagi, "The Express-Net : A Local Area Communication Network Integrating Voice and Data," Proc. Int. Conf. Perf. Data Commun. Syst., Paris, Sept. 1981.

[GMMS88] M. G. Gouda, N. F. Maxemchuk, U. Mukeherji, and K. Sabnani, "Delivery and Discrimination: The Seine Protocol," AT&T Bell Laboratories Research Report.

[GS80] R. L. Graham, N. J. A. Sloane, "Lower Bounds for Constant Weight Codes," IEEE Trans. Information Theory, Vol. IT-26, No. 1, Jan. 1980, pp. 37-43.

[Ko78] Z. Kohavi, *Switching and Finite Automata Theory,* McGraw-Hill, 1978.

[Ma87a] N. F. Maxemchuk, "Random Access Strategies for Fiber-Optic Networks," Infocom '87, San Francisco, Ca., Mar. 31 - Apr. 2, 1987, pp. 307-311.

[Ma87b] N. F. Maxemchuk, "Twelve Random Access Strategies for Fiber-Optic Networks," Submitted for Publication.

[MB76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun. ACM, vol. 19, pp. 395-404, July 1976.

[MMWS88] U. Mukherji, N. F. Maxemchuk, C. W. Wu, and J. C. Swartzwelder, "Transmission Format and Receiver Logic for Random Access Strategies in a Fiber-Optic Network," IEEE Computer Networking Symp, Washington, D.C., April 11-13, 1988.

[Mo83] D. J. Morris, *Pulse Code Formats for Fiber Optical Data Communication, Basic Principles and Applications*, Marcel Dekker, New York, 1983.

[TC83] C.-W. Tseng, B.-U. Chen, "D-Net, A New Scheme for High Data Rate Optical Local Area Networks," IEEE J. Sel. Areas in Commun., vol. SAC-1, No. 3, April 1983, pp. 493-499.