A COMMUNICATION PROTOCOL FOR DISTRIBUTED MICROPROCESSORS

M.S. Sloman*, B.K. Penney*, T.A. Marsland†

*Department of Computing and Control, Imperial College, London SW7 2BZ.
†Computing Science Department, University of Alberta, Edmonton, Canada.

INTRODUCTION

With advances in LSI technology it is now practical to incorporate cheap microcomputers directly into equipment being remotely controlled. This has made economic the possibility of implementing distributed control where each processor manages a local function and communicates with the other processors in order to coordinate actions and achieve overall cooperation.

A microprocessor based communications system can also provide cost savings in the traditional centralised control environment in which control and monitoring lines are brought to a central computer (Oswald (1)). The amount of information transmitted from the point at which it is generated can be reduced by partially analysing and formatting the data, and possibly using data compression. In addition the microcomputer's memory can act as a buffer to smooth out the 'bursty' nature of data transmission. This will result in lower bandwidth requirements and may permit the use of serial rather than parallel transmission lines. Distributed systems of this type require very reliable communication between processors, so one purpose of this paper is to present a protocol for handling the message exchange.

Standardised communications protocols are required which meet the requirements of: *Simplicity* - the protocol should be easy to implement on a microprocessor with its limited memory and processing power; *Flexibility* - the protocol should be independent of the transmission methods and network configuration; and *Reliability* - industrial environments are likely to be electrically noisy, necessitating automatic error detection and correction techniques.

NETWORK CONFIGURATION.

Networks fall into two fundamental classes: *broadcast or bus* systems, in which all nodes communicate directly via a common transmission medium, and *store and forward* (S & F) systems where pairs of nodes exchange messages through an arbitrary number of intervening nodes. The latter organization includes partially connected, star, tree-structured and some loop networks (Casaglia (2)). The transmission medium consists of several point-to-point data links, many of which may be active at the same time thus increasing the effective network throughput to the sum of the capacities of the active links. In partially connected networks alternative paths may exist between any pair of nodes, and so each one must provide adequate buffer space for in-transit messages and be capable of re-routing the transmission.

In broadcast systems the medium (eg. radio, coaxial cable, light pipe) is used in such a way that a signal emanating from one node is received almost simultaneously by all others. After initial processing the signal is ignored by all stations except the one addressed. No routing problems arise since communication is direct to the destination, but contention (multiple nodes trying to transmit at the same time) must be resolved as in the Ethernet of Metcalfe and Boggs (3). Contention may arise in point to point half-duplex systems. Since there are no duplicate paths, and no storage within the network, the common transmission medium must have higher bandwidth than a typical point-to-point link. Not only does this entail the use of more expensive high-speed components, but also the communication line may have to be duplicated for sufficient reliability. In Table 1 the properties of the two configuration classes are summarized. In a hierarchic control system a hybrid configuration such as multiple buses connected by S & F nodes might be appropriate.

STRUCTURE OF A COMMUNICATIONS SYSTEM.

The communications system consists of a number of independent levels with a protocol defining rules for sending messages between the corresponding levels in different nodes, and a specified interface between adjacent levels in the same node.

The user level constitutes the application dependent processes which communicate with each other by calling a message handler. This provides a uniform interface for exchanging messages with local or remote processes. It is necessary to limit the length of a message in order not to delay other nodes from access to the transmission medium, particularly in a broadcast system, and to reduce the buffer size requirements at the intermediate nodes in a S & F system. The communications system could split long messages into smaller packets for transmission but, in order to reduce the complexity and prevent reassembly problems, it is simpler to limit the maximum length of a message to that of a packet. The user level interface is described later, but the user level protocol is not discussed since it is application dependent.

The message handling level performs the following functions:

- Generates and strips off packet headers;
- Maps between user level process names and communication system addresses;
- Manages buffer allocation;
- Handles timeouts if they are required at the user level;
- Manages the routing of packets;

In order to control the flow of packets round a network it is necessary to prefix the data portion with a short header, which is used to identify the nature and ultimate estination of the packet. Incoming packets rill be passed up to the user process unless hey are in transit to another node. The outing algorithms can be quite complex and re amply discussed in Davies and Barber (4).

he communication link level is concerned ith the exchange of packets across a data ink. It is responsible for error correction, ontention resolution and packet acknowledge- ent. In a broadcast system the node address ield within a packet must be inspected in rder to determine whether to receive the acket. Some of the functions of this level ould be implemented in hardware, e.g. ddress recognition.

he transmission level handles bit and haracter synchronization, the signalling ethod, contention and error detection, and o is line technology dependent.

OMMUNICATION SYSTEM REQUIREMENTS

ddressing Capabilities

n order to cater for various network onfigurations, every packet must identify oth source and destination node addresses. n a S & F network the addresses are edundant at the communication level, but re required by the routing function. In any communication systems it is desirable o send the same message to many different estinations. These *scatter messages* either ropagate through the system by retransmission, carry a unique destination address which s accepted by all nodes, and so should not acknowledged.

rror Detection and Correction

he protocol must be able to handle the ollowing errors.

nformation corruption due to interference. n electrically noisy environments random nd burst errors may appear in the packet. hese errors are effectively detected with cyclic redundancy check (CRC) (Martin (5)) nd their correction is possible through he use of the automatic repetition (ARQ) ethod (5). Upon arrival of each error-free acket, the receiver returns a positive cknowledgement. If the sender does not eceive the acknowledgement within a specified imeout period the packet is sent again. lthough a negative acknowledgement could educe the timeout period it increases the verhead traffic and requires the recognition f an extra packet type.

oss of packet. A packet may not reach its estination because the line or receiving ode is temporarily out of action, or the estination address is corrupted. The ARQ ethod accommodates these situations by nsuring that the sender retransmits the acket.

acket incomplete. Once the transmission of packet has started, the data must reach the ext node in a finite time. The end of the acket is located either from the length ield as in DDCMP (6), or via a special end f packet flag as in HDLC or the bipolar iolation technique used in the Newhall loop 7). However, the flag may become corrupted, r the sender may abort the message or specify he wrong length. Thus the receiver must also nitiate a timeout at the start of a packet nd, when failure to complete a transmission is recognized, should revert to packet synchronization mode.

Packet sequence error. Should an acknow- ledgement be lost, or not arrive on time, the packet is sent again. All packets must there- fore contain a sequence or identification number, so that the receiver can detect duplicates. A single bit sequence number (toggled for consecutive packets) is adequate if the number of outstanding unacknowledged packets is limited to one. The acknowledge- ment contains the sequence number of the packet to which it corresponds, so those duplicates are also recognised.

Design Goals.

In order to *prevent congestion*, since the buffer space is limited at every node, the rate of transmission must be controlled. This can be achieved by limiting the number of outstanding unacknowledged messages. In the simplest case only one packet at a time is outstanding.

The communication protocol should be *transparent* to the information being sent, and so must not rely on reserved control symbols. This conflicts with the require- ment for a specific bit sequence to achieve bit and character synchronization. For serial transmission the bit insertion technique can be used, to prevent the synchronization flag (01111110) from occurr- ing in the packet (8). On transmission a "0" is inserted after all sequences of 5 contiguous "1" bits. On receiving, a "0" which follows 5 contiguous "1" bits is discarded. Any detected sequence of six "1" bits is the flag or an error. This technique is not applicable to parallel transmission and an alternative is to specify the length of the packet in the header.

The various levels in the communication system should be *independent*, allowing net- work configuration and transmission methods to be chosen to suit a particular applica- tion. In addition the protocol should be suitable for both hierarchic and linear structures.

Application Dependencies.

In process control applications it may be necessary to indicate *alarm conditions* to a central control room. These messages must get through ahead of normal traffic so some priority scheme is required. Of equal importance is the need for a short *response time* to a request. Since responses are delayed by other traffic on the network any technique which reduces that traffic is use- ful. One method is to combine an acknow- ledgement with a data packet, when they simultaneously want to use the same link.

EXISTING PROTOCOLS

Some of the commercially available data link protocols are ISO's HDLC, (now part of CCITT X25 protocol (8)), IBM's SDLC and DEC's DDCMP. All of these are designed for point- to-point communication or, in the case of SDLC and DDCMP, multipoint with a single master which polls slave nodes. The packet headers do not allow for both source and destination addresses which are essential for broadcast systems. In addition, DDCMP is the only one which specifies a length field in the packet header, the others rely on bit insertion to ensure that the start and end of packet flags are unique.

The X25 protocol uses the concept of a virtual call between two host (user level) processors i.e. a point to point circuit which may traverse several nodes. It is not necessary to specify source and destination addresses as these can be obtained from the virtual call number, but the nodes then have the overhead of translating the call number into actual addresses. There is also the overhead of the messages required to set up and clear a virtual call for a short transaction, which in a control application may consist of only two messages, a request and a reply.

In general the above mentioned protocols are far too complex with many unnecessary message types, not needed in a local network of co-operating microprocessors.

COMMUNICATION SYSTEM PROPOSAL.

Our protocol is specifically designed for industrial applications, where the probability of a transmission line error is relatively high. On these lines use of the ARQ method with its positive acknowledgements is recommended. At the message and user levels, the probability of failure is much lower and use of a negative acknowledgement in the form of an error message is proposed. This technique has the advantage of reducing the overhead traffic, while ensuring that notification is received of all permanent errors. Note that in many cases loss of a single message is of little consequence, for example, in the logging of stochastic data.

One advantage of the broadcast system is that the communication level is sufficient to guarantee error free end-to-end communication. In contrast two classes of error are possible in store and forward networks.

(i) The originating process is advised of an error when, in fact, the transmission was successful. For example, a communication level acknowledgement was never received, even though the packet was forwarded.

(ii) The error message cannot get through, or a node fails after sending a communication level acknowledgement, but before forwarding the packet, and so cannot generate an error message.

These problems can only be overcome by generating a positive end-to-end acknowledgement for every packet. This increases the response time over-head in the case where the probability of successful transmission is very high. Rather than implementing automatic acknowledgements at the message level, we recognize that this can be done more selectively by the user. For example, when the original message is a request for information, the user level reply can serve as the end-to-end acknowledgement. Should the destination process be unable to supply the requested information immediately, it would acknowledge with an empty packet and send the data later.

Communications Level Protocol

An illustration of the format of the packet is given in Fig. 1a. The transmission level detects the start of packet, assembles the header and checks for a CRC error. If one is detected the packet is ignored, else the header portion is passed up to the communications level for immediate processing. Meanwhile any data portion is being assembled and checked. The communication level recognises two types of packet and uses the link control field shown in Fig. 1b to process the packets according to the simplified state diagram shown in Fig. 2. An information packet (I=1) contains information for the message or user levels. It carries a transit sequence number (T) which is toggled for each new packet and is compared with the expected transit number (ET) at the receiving end. Table 2 indicates the action taken on receiving a packet. When an information packet is acknowledged the received T sequence bit is transferred to the response sequence field (R) and the information acknowledge bit (IA) is set in the outgoing header.

Where possible this acknowledgement is combined with a new message awaiting transmission. When a packet arrives containing a valid acknowledgement, IA=1 and R = transit number (LT) of the last message sent, the buffers for the acknowledged message are released.

Command packets (C = 1) are used to set the communications level, and possibly the whole communications system in a node, into a defined state. These packets must always be accepted, since they may reset sequence numbers, but are designed so that it does not matter if they are duplicated. The command function specified in the F field is performed only at the final destination although some functions would not normally be forwarded. Commands are acknowledged by setting CA = 1, and only one can be outstanding at a time. Obviously the C and I bits cannot both be set, but might both be zero. Four functions have been specified, but others might be required for monitoring, diagnostics, and status checking.

Reset - Reinitialises communication between two nodes, typically establishes sequence number agreement along a communication line.

Bootstrap - Forces the node into bootstrap mode. The data portion of the packet contains the object code and load address of the program being sent. Typically this will be a new message level handler.

Initialise - Reinitialises all buffers, sequence and identification numbers, etc., in the communication system. All outstanding messages will be lost.

Start - Is the only command which will be executed after a *Bootstrap* or *Initialise* have been processed. No other packet can be accepted or acknowledged, even if intended for another node. This command allows controlled restart of part of the network.

Message Level Protocol

The details of the message level control fields are given in Fig. 1c. This level recognizes four message types, all of which are information packets at the communication level.

Information messages are passed on to the user level as they either contain data or replies. The four FUNCTION bits specify: a reply expected, so a timeout is initiated; the message is allowed to generate an error; the message contains data and so the TID field is valid; the message is a reply and so the RID field is valid.

Error messages are sent by an intermediate node but appear to come from the intended destination and have RID = TID of the abandoned message. The nature of the error is described in the function field. These cannot generate other error messages or expect replies, otherwise a fault could propagate an endless series of error messages.

Command messages are for the message level of the destination node and are not passed up to the user level. They serve a similar purpose to the commands at the communications level and so contain no identification numbers. Typical command functions would be: bootstrapping of the user level; resetting message level identification numbers; setting routing tables; loop back testing.

Command acknowledge indicates that the command message was successfully received by the destination. No other messages will be transmitted to the particular destination between transmitting a command and receiving the acknowledgement.

The PRIORITY field is used to indicate alarm messages which are allowed to jump the queue of waiting messages, but this is incompatible with maintaining the correct sequencing of messages. One solution would be to maintain different sets of sequence numbers for each priority level to each possible destination, but this would increase the complexity and storage requirements. The alternative is not to implement sequence checking at the message level but to provide the facilities for this to be incorporated in a user level protocol if required. When the message level receives a message from the user level, a TID No. is assigned by incrementing the last one used for that particular destination. This TID is passed to both the source and destination user levels. If the reply expected bit is set, the message level starts a timeout which is cancelled by a reply or error message with the matching RID. The user level receives an error indication if the timeout fails and so it could retransmit the message but then it will be assigned a new TID. It is recommended that only one reply from a particular destination should be outstanding at any time. In those systems where the route is preset the communications level is adequate to ensure that messages of the same priority arrive in order.

User Level Interface.

The precise requirements for a protocol at the user level are likely to be application dependent. We have, therefore, provided two primitives SEND and RECEIVE which can be used to implement a wide range of protocols for the exchange of messages between processes. The precise behaviour of these primitives is specified by parameters which are set by the user level on call or provided by default. Some of the parameters are set by the message level on return.

The SEND parameter list consists of :

Destination - is the name of the destination process, which could be in the same node. It is translated by the message level into the DEST address in the packet header.

Message - is a pointer to the body of the message.

Length - the message length in bytes.

Priority - message priority.

Error Generation - a boolean which signifies whether the user level should be notified via an error message if the communication system is unable to deliver the message.

Reply request - a boolean signifying whether the user level expects a reply. The message level initiates the specified timeout.

Timeout - the time allowed for reply to a message before the message level generates an error. It is dependent on the round trip delay. If a reply is received after a timeout expires it would still be passed to the user level which can decide what to do with it.

The SEND call on the message handler will return the TID No. assigned to the message, or an error indication if it was unable to accept the message (e.g. no buffer space). The TID is used to identify the reply.

The RECEIVE primitive does not have any call parameters, the highest priority message will be picked up. The return parameters are as follows :

Sender - This is either the name of the sending process or an indication that no messages are waiting. It is not necessarily the same as the *Source* address field in the packet header.

Message - The pointer to the next message, if any are present.

Identification - The TID and RID numbers of the received message.

Length - the message length in bytes.

Reply expected - a boolean signifying whether the sender expects a reply i.e. has initiated a timeout.

The decision on whether interrupts or polling should be used to indicate the arrival of a message is application dependent because the control process might be time critical and so should not be interrupted, see Goldsack (9).

In many applications it would be desirable to provide a further layer of application dependent software to simplify the interface to the message level. It could also provide additional functions such as waiting for a message from a particular sender.

CONCLUSIONS

Some of the design decisions which have to be made in the implementation of an efficient communications protocol for distributed control applications have been discussed. It has been shown that while store and forward communications increase the complexity of the software, they can provide low-cost alternative paths for increased system reliability. Broadcast networks, on the other hand, although more expensive in hardware costs, allow simpler software organisation and do not produce such serious routing and error recovery problems.

Although a number of protocols have been developed by industry and research establishments (6,10,11) these place restrictions on the configuration, or are not suitable for networks of cooperating microprocessors for other reasons. The advantages of the protocol described here are as follows :

i) A positive acknowledgement is used at the data link level, which is assumed to be prone to burst noise. Under these conditions use of an automatic retransmission is preferable to the continuous overhead of an error correcting code.

ii) Faults detected by higher levels occur less frequently but are likely to be permanent, so user processes are advised of such a condition through a negative acknowledgement in the form of an error message.

iii) Elimination of automatic end-to-end acknowledgements and the method of combining acknowledgement and the data packets reduce the overhead traffic. This improves the network response time.

iv) The protocol is applicable to both store-and-forward and broadcast networks, and so can also be used in hybrid systems.

v) Use of a single-bit transit/response sequence number considerably simplifies the communication level software. Increasing the range of these sequence numbers would allow the detection of spurious packets, but this would increase the software cost.

Although the actual communication system remains transparent to the user, he still has the responsibility for explicit end-to-end acknowledgements and decomposing long messages into packets. Overall, this protocol allows efficient exchange of messages between microcomputers, for both broadcast and store and forward networks. This could lead to standardisation of the protocol and the development of suitable LSI circuits allowing 'black box' design of communications systems for a wide range of distributed control applications.

REFERENCES

1. Oswald, H. 1976, INFOTECH Distributed Systems, 363-374.
2. Casaglia, G.F. 1976, Euromicro 2, No.4, 5-18.
3. Metcalfe, R.M. and Boggs, D.R. 1976, Comm. ACM, 19, 395-404.
4. Davies, D.W. and Barber, D.L.A. 1973, "Communication Networks for Computers" Wiley.
5. Martin J. 1970, "Teleprocessing Network Organization", Prentice Hall.
6. Digital Equipment Corp. 1974, "Specification for digital communications message protocol (DDCMP)".
7. Farmer, W.D. and Newhall, E.E. 1969, Proc. ACM symp. Problem Solving in the Optimization of data comm. systems., 1-33.
8. CCITT 1975, "Recommendation X.25", COM VII, No. 262E.
9. Goldsack, S.J. 1977, This Conference.
10. Daresbury Laboratory Network Protocol, privately circulated.
11. Liu, M.T. and Reames, C.C. 1977, 4th. Annual Symposium on Computer Architecture, ACM-SIGARCH 5, No.2 193-200.

BROADCAST	STORE AND FORWARD
No routing hence simpler software.	The need for routing and end-to-end acknowledgements adds to the software complexity.
High-speed line needed to provide capacity for all communications. Line interface is likely to form a large proportion of the cost of a node.	Multiple communications can take place simultaneously over point to point lines - low speed standard LSI interfaces can be used.
Redundant communication paths are expensive.	Alternative communication paths easily provided.
Transmission path is via a medium which is passive or has simple regenerators and hence is comparatively reliable.	Transmission path is via store and forward nodes, which may be less reliable due to their additional complexity.
Usually a single half duplex line to a node.	Generally multiple lines to each node.

TABLE 1 - Summary of Basic Network Properties

I = 1

$T \neq ET$:	Duplicate packet so discard, but acknowledge with R = T received.
$T = ET$:	New packet received so acknowledge with R = T received, pass up to message level, toggle ET.

IA = 1

$R \neq LT$:	Duplicate acknowledgement so ignore.
$R = LT$:	Successful acknowledgement so release buffer, ready to transmit next packet.

Table 2 - Actions performed on receipt of an information packet.

SYNCH. PREAMBLE	DEST ADDR	COMMS. LEVEL CONTROL	SOURCE ADDR	MESSAGE LEVEL CONTROL	LENGTH	HEADER CRC	DATA	DATA CRC
≥8	8	8	8	16	8	16	variable	16 bits

Figure 1a Packet Format

C	CA	IA	F			T	R
			I	-	-		

C - Command packet

CA - Command acknowledgement

I - Information packet

IA - Information acknowledgement

T - Transit number

R Response number

F - Command function

Figure 1b Communications Level Control Field

2	2	4 bits
PRIORITY	TYPE	FUNCTION
TID		RID

PRIORITY - is used to order the queue of outstanding packets, and it could be used by the communications level to pre-empt an existing transmission.

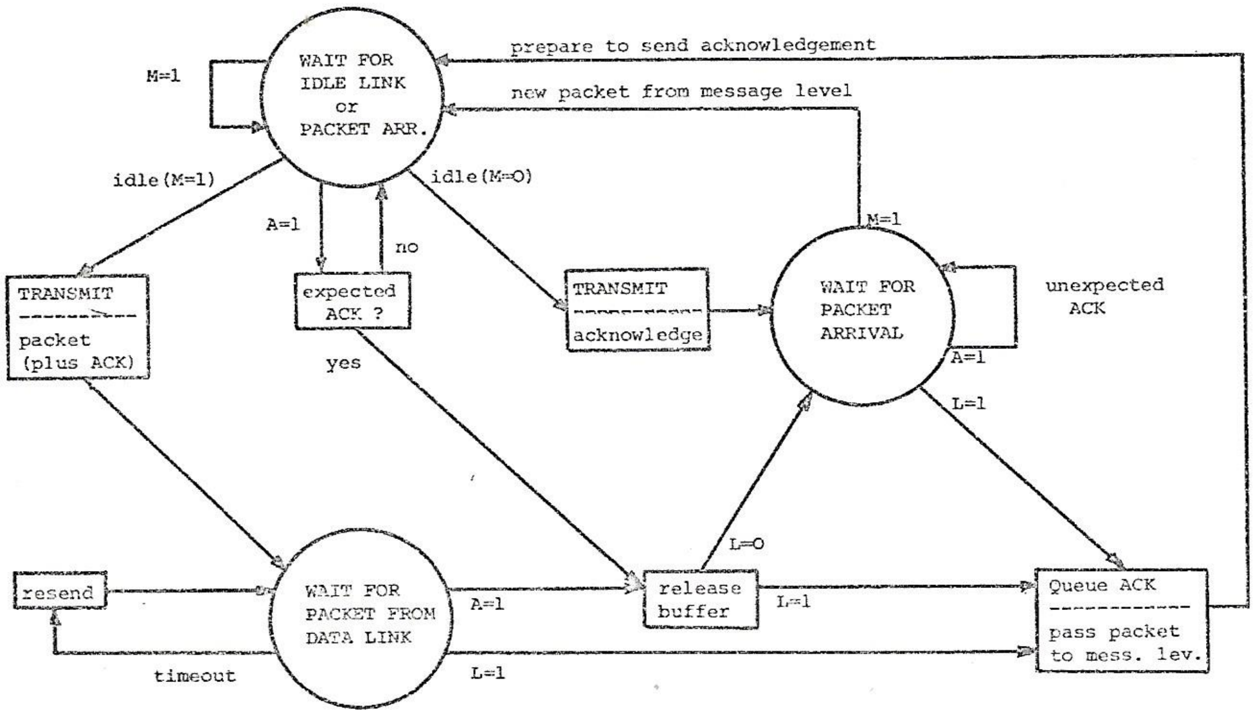
TYPE - the message type.

FUNCTION - additional bits whose interpretation depends on the message type.

TID - the identification number of the message being transmitted.

RID - the response number used to cancel timeouts and identify replies.

Figure 1c Message Level Control Field


WAIT FOR IDLE LINK or PACKET ARR.
M=1
prepare to send acknowledgement
new packet from message level
idle(M=1)
idle(M=0)
A=1
no
yes
TRANSMIT packet (plus ACK)
expected ACK ?
TRANSMIT acknowledge
WAIT FOR PACKET ARRIVAL
M=1
unexpected ACK
A=1
L=1
L=0
resend
WAIT FOR PACKET FROM DATA LINK
A=1
release buffer
L=1
Queue ACK pass packet to mess. lev.
timeout
L=1

L = 1 - Received packet from data link
A = 1 - Packet contains an acknowledgement
M = 1 - New packet from message level for transmission

Figure 2 Simplified Communication Level Transition Diagram.