An integrated approach to network protocols

by LOUIS POUZIN

Institut de Recherche d'Informatique et d'Automatique
Rocquencourt, France

INTRODUCTION

Host-to-host protocols (H-H) for heterogeneous computer networks are still in infancy. So far very few implementations are in existence. Among those on which documentation is available are Arpanet1 and Cyclades.2 The former provides only for basic services allowing the transfer of up to 1000 octet messages, with flow control but not error control. The latter allows up to 32 000 octet messages, with error control. Both are similar in the sense that they offer only a message transfer service, which is intended for building higher level protocols more appropriate for specific uses. Since data to be transferred are usually structured in various ways, a traditional approach is to superimpose additional layers of specific protocols, each one dealing with a particular level of structure. While being functionally correct, this approach leads to heterogeneity, redundancy and overhead among the various layers.

Part of the difficulty can be attributed to the heterogeneity of operating systems, for which there is no well accepted common model. Another reason is the mistiness of the host concept, which is usually assumed to be of a certain kind by protocol designers. There is no doubt that a big enough computer can support a protocol of several thousands of instructions. But what if a host is simply a terminal? It should be kept in mind that communication between an application program and a terminal must wind its way through an H-H protocol somewhere along the path. If this protocol logic is rather involved, it will be economically necessary to share it between a number of terminals. Hence the need for concentrators, which may be considered as mini-hosts, (Figure 1). But is this the only desirable solution?

TERMINAL ACCESS

Most existing networks are actually star networks of terminals. In the coming years they will tend to adopt packet switching, and at the same time introduce more efficient and homogeneous end-to-end protocols (E-E) no longer tied up with the intricacies of the communication gear. Application programs and terminals will become independent of communication systems.

By integrating several overlapping star networks there will be a need for computer-to-computer communications. But the most frequently required type of communications will be terminal access. This trend is consistently pointed out by various forecasts, such as Eurodata. Consequently, the development of networks will be strongly influenced by the overall cost of accessing terminals, including all communications gear interposed between computers and terminals.

Minimizing the cost of terminal access includes reducing line mileage and bandwidth, as well as intermediate intelligence necessary to relay communications. The minimum logic required by a terminal is a transmission procedure, if the traffic is to be sufficiently reliable. Line mileage and bandwidth are reduced with multi-point local networks using packet mode traffic. Loop3 or tree networks4 make terminals accessible through a ramification of inexpensive links relayed by packet concentrators and base-band modems. This should be more economical and flexible than traditional multiplexing or concentration, (Figure 2).

At this point, one may wonder what is a host. Any terminal could be a host, but this is not very economical in terms of address space, since the address field would have to be tailored for a much larger population than is usually anticipated for hosts. Another approach would be to interpose a mini-computer as a front-end to the communication network for the sake of playing host to the terminal cluster, (Figure 3). This bogus host could indeed perform its part of an H-H protocol, but this is not basically different from the traditional concentrator approach. Since its presence does not appear a necessity from a communication standpoint, why not do away with it altogether?

DISTRIBUTED HOST

Communications between hosts are not an objective per se. What is actually aimed at are communications between entities, which represent a meaningful set of self-contained correspondents, e.g. processes, files, devices, jobs, subsystems, etc. Since many of these entities are usually housed and activated simultaneously on a single host computer, it is customary to refer to an H-H protocol while what is meant is actually the set of communication rules between corresponding entities in different hosts.

For generality, communication rules tend to be inde-
dependent of the nature of the corresponding entities, at least for a class of basic services. Therefore, entities are not specified in the definition of protocols. Instead, there appear anchoring points called ports, to which they are bound at communication time.

Although they abide by the same protocol, communications between pairs of ports are logically independent from one another. However, they must share some resources, like an I-O device, a telecommunications package, a communication network. Therefore, protocols must include the information necessary to multiplex shared resources in an orderly manner, so as to preserve logical independence. On the other hand, there is no point in making it a requirement that ports belong, somehow to certain host computers. Whether they happen to share the same computer or not is immaterial as far as communication protocols are concerned. A host address is no more than an information required for the multiplexing of some resources. It does not have to be a particular computer.

For example, in the network structure of Figure 2, the concept of host reduces to a set of transmission lines and packet concentrators, which are a shared resource bearing a host name.

Such a construction may be termed a distributed host. One may notice that in this example the technology of packet concentrators is not at stake. They may be programmed mini-computers or hard-wired delay registers. The same concept of distributed host would hold.

It should be clear by now that an H-H protocol on a distributed host cannot be implemented on a particular computer. Furthermore it may well be that nowhere can be found enough computing power to implement a protocol of a few thousands instructions. As a result, only simple and compact protocols are acceptable.

PORT-PORT PROTOCOL

Since a host may not represent a geographically well defined destination, there should be literally no H-H protocol. When two ports communicate, there should not be any logical interference from other ports, hosts, or communications network. In other words, all the machinery interposed between ports should be transparent. E.g., protocols used at inner levels should remain invisible. Port-Port protocols are just end-to-end (E-E) protocols.

In order to facilitate communications in a heterogeneous environment of hosts and terminals, it would be desirable to adopt existing or proposed standards. The trouble is that no E-E protocol suitable for networks has yet been advanced in standardization bodies such as ISO or ECMA. HDLC is only valid for wire-like transmission media.

One can vaticinate that HDLC with double numbering scheme will be a de facto standard at some point in the near future, since it is already introduced by IBM in its new products lines. Although HDLC is actually designed only for the control of a physical data link, it may well serve as a starting point for the design of a network E-E protocol. This approach may result from sheer inertia, or mimicry, or IBM pressure on the market. Another reason is that HDLC contains some basic features reasonably close to what might be required for an E-E protocol. Anyhow it seems worth attempting to define an E-E protocol applicable to communications between ports by borrowing from HDLC. At least that may sound as a sensible enough move on the standardization scene.

PROTOCOL TRANSPARENCY

Protocols between ports are only sub-functions invoked by higher level logic for the purpose of transferring information from one domain to another. This transfer action is supposed to remain transparent in terms of information contents. In other words, no alteration should occur. Actually, no physical communication medium can be assumed error-free. Thus mechanisms for error detection and recovery are usually associated with E-E protocols in order to lower the error rate down to a required level. However there remains always some residual probability of error, which may be considered as acceptable, or negligible, for the purpose at hand.

Indeed higher level constructions may contain additional mechanisms which are able to cope with E-E pro-
tocol errors, and make them disappear for all practical purposes. In other words, E-E protocols are designed to be normally transparent, except for residual errors, which may even be detected, but not necessarily recovered. What are the nature and the rate of acceptable errors is a matter of economics in distributing properly control functions across levels of protocols.

HIGH LEVEL PROTOCOLS

Transferring information is only a tool toward the achievement of some practical goals, e.g., interaction between process and terminal, or file transfer. These are typical examples of high level functions which are so commonly used in networks that they are worth packaging in some standard form. They also have come to be called protocols in the network literature, such as virtual terminal protocol, or file transfer protocol.

A high level protocol such as file transfer may involve more than two hosts. It is no longer a simple E-E protocol. Actually a high level protocol is a set of rules defining the working of a distributed machine, which is designed for handling a particular application. One may notice that this definition applies to E-E protocols as well.

In order to perform its task a distributed machine must accept commands, input or output data, and transfer information between its components. These exchanges may carry data, or commands and state variables necessary for the synchronization of the whole machine.

While data can be altered intentionally, as part of the machine task, (e.g. data conversion or reformatting), communication mechanisms between the distributed components should be ideally transparent. Thus E-E protocols may be used as building blocks to carry out any information transfer required by a high level protocol. This makes for strong incentives to define general and efficient E-E protocols. Otherwise the piling up of layers of protocols would ultimately result in excessive overhead.

PROTOCOL NESTING

A unit of information, say a message, exchanged between a terminal and a process may be restricted in length by mutual agreement. But this is not necessarily short enough to comply with the characteristics of a communication network, which may use smaller units for various reasons of cost-effectiveness. Therefore messages to be transmitted must be fragmented, and the fragments must be reassembled at the destination.

Similarly, transferring a file cannot usually be accomplished by just sending one message. The file must be fragmented in blocks, pages, records, which are sent individually, and put back together onto the destination file store.

Sending a job to a distant computer may involve the transfer of a set of files. More generally transferring a unit of structured data requires breaking it into a set of pieces, which are again broken down repeatedly until one reaches a level of fragmentation suitable for physical transfer, (Figure 4). This scheme is general, and does not make any assumption about the physical or the logical structure of data.

At each level of nesting there appear similar functions:

- break logical unit into fragments
- transfer fragments separately
- put fragments back into logical unit

This is again repeated, as each fragment becomes a logical unit and is broken down in sub-fragments, etc... up until no more fragmentation is necessary. A way of representing this process is traversing a tree structure, (Figure 5).

A first idea that comes immediately to the mind is that transferring information could well be a recursive application of the same protocol.

Let us call \( S(N,F) \) the function called upon at the \( N \)th level of fragmentation for sending the fragment \( F \). The function \( S(N,F) \) may be expressed as follows:

**sender:** procedure \( S(N,F) \)
begin if fragmentation required then
begin make fragments \( f(1) \ldots (p) \);
for \( i := 1 \) step 1 until \( p \) do \( S(N+1,f(i)) \)
end
else send \( F \)
end

The corresponding function \( R(N,F) \) at the other end
might be expressed as follows:

receiver: procedure $R(N,F)$
begin if fragmentation then
  while fragment missing until time-out do
    begin $R(N+1,f)$;
      if $f$ good then place $f$ in $F$
    else drop $f$
  end
  else receive $F$;
  if $F$ bad or time-out then report $F$ bad
end

These algorithms are obviously a first approximation. They contain a limited set of functions necessary to carry out communications. It may now be appropriate to examine what differences or additional requirements might appear in practice.

**Error recovery**

The previous algorithms detect errors and report them to the next higher level of control. Ultimately the highest level gets the report and starts all over again. But this may not be satisfactory in terms of efficiency. Transit delays and possibly transmission overhead are reduced when error recovery is performed at lower levels, on small fragments.

E.g., one might introduce additional functions whereby the receiver sends back an acknowledgment for each good or duplicate fragment received. At the other end, a repetition process sends again any fragment not acknowledged within a certain delay. As can be predicted, this error control scheme induces overhead of its own in transmission (ACK information) and in processing.

Furthermore, handling error recovery at a particular level does not make it error free for all higher levels, because it cannot be guaranteed to be always successful. Consequently, the highest level must always be prepared to get an error report. Recovery at lower levels can only reduce the error rate down to an acceptable figure.

There is no simple criterion to help determine at which level error recovery would be the most effective. First, error control overhead in transmission, processing, or buffering depends on the many variants in acknowledgment schemes. Second, error and traffic patterns, user constraints, system characteristics, are also major factors bearing on error control effectiveness.

Therefore, one may say that error recovery should appear at some levels of protocols, depending on the environment. A convenient way is to make it an option, both in time, and at a particular level.

**Flow control**

Resources necessary for transferring information must be made available, on the spot, or by prior reservation. Allocation schemes may be desirable to prevent traffic un-stability and interferences between independent correspondents.

Again flow control adds up its own overhead, and it is likely not desirable to place it at all times at all levels. This is also a feature which one would like to turn on and off at a particular level, depending on the environment.

**Fragment identification**

A noticeable difference in handling fragments at the receiver end derives from the sequence of arrivals. When the sending order is normally maintained by the communication medium, any fragment can be identified by its order of arrival. Since errors may occur, fragments carry a sequence number, and any fragment arriving out of sequence is an indication of error. At this point recovery may take over, at some level of protocol.

On the other hand, when fragments may normally arrive out of sequence, they should not be rejected, except if this occurrence is practically negligible. Thus some buffering is needed to park temporarily early fragments waiting for logical predecessors. In the algorithms given previously, there is no assumption about the arrival sequence, as long as all fragments belong to the same next higher level unit. Otherwise, two alternatives may be considered:

(a) fragments of the next unit are rejected. They may be kept from being sent by synchronizing the next higher level of protocol at unit level.
(b) fragments belonging to a future unit are just stored until the corresponding instance of the protocol is activated.

It is clear that fragments must be labeled so that such an identification can take place.

**Other differences**

The semantics of the algorithms differ depending on the physical and logical structure of fragments. E.g., reassembling pieces of a file on disk is not the same as for a message in core memory. Reporting a failure may take different forms. Time-out’s are adjustable parameters. System protection and access methods may introduce quite a number of differences in the practical handling of fragments.

For all sorts of reasons mentioned previously, it may not be practical to use identical routines or algorithms to implement the various levels of protocols. However, similarities are substantial enough to warrant some effort toward a general scheme applicable at all levels.

**Control information**

In order to handle each fragment properly, some additional information is necessary for fragment identification and to allow each end of the protocol to work in synchroni-
zation. This is usually packed into a header preceding the text or the descriptor of the fragment proper.

When nesting protocols a classical method is that each level wraps every fragment in its own envelop. If further fragmentation is to occur, the complete fragment (header and text) is broken into smaller pieces, each of which receives a new header sticked by the nested protocol. This works of course, but each level of header increases the amount of transmission overhead. If each level of fragmentation produces statistically a large number of smaller fragments, then headers at intermediate levels contribute a relatively small part of the total overhead. But nesting does not always produce many fragments, since it may be required only to pass through a particular layer of the system hierarchy. Then each level of wrapping duplicates to a large extent control information in every nested header.

In compensation for overhead there is the advantage of allowing the design of each level independently. For lower levels, fragments are just unrelated pieces of information. This approach is indeed desirable when it is essential to avoid unnecessary coupling between layers of systems, e.g., when some layers are shared by unrelated users.

On the other hand when system-wide consistency is desirable for efficiency, and in the domain of realistic objectives, one may attempt to define a common set of control information intended to be used by every layer of protocols. This may reduce effort in design, help standardize well defined protocols, and ease out error recovery, since every fragment may be directly related to a logical higher level unit of information.

NESTED PROTOCOL CONTROL

When implementing a full-fledged protocol at some level of communications, it is possible to identify two classes of control information exchanged between sender and receiver. One class is associated with the direct handling of the text part of messages, in order to make sure it is correct, and to locate it in relationship with other text parts. Another class pertains to the specific mechanisms intended for the synchronization of the two ends of the protocol.

In a nested environment, the first class may be called global variables, since the text is independent of the protocol level. The second class constitutes local variables which are part of the execution context of the protocol. They should not normally appear at other levels, except for those variables used as run time arguments when control passes up and down across levels of nesting.

Actually, one may often identify a third class, such as "piggy-back" information. But this is in essence independent of the protocol. Only messages are used as ready to go containers for hitch-hiking information that belongs to another protocol, i.e., the sender-receiver pair dealing with traffic in the reverse direction.

In the following we shall examine what control information could make up a common header at all levels of nesting. The objective is that each level of protocol should only modify certain fields of a fragment header, rather than adding its own. When fragmentation occurs, each sub-fragment receives a similar header, with only changes in certain fields. There are also suggestions for field sizes.

P-level (4 bits)

This is a number indicating the nested level of the protocol in charge of processing this message. (Higher numbers are lower levels of protocols; they may be considered as negative if one wants to keep consistency in terminology.)

U-NR (7 bits)

This is the identification number of the logical unit being fragmented or reassembled at this level of protocol. It is also a fragment number passed as argument between this level and the next higher. Its use is to segregate fragments of other units arriving out of order.

F-NR (7 bits)

This is the fragment identification number of the text part carried within this message. It becomes a U-NR passed to the next level down when further fragmentation is required. It may be used circularly when the number of fragments exceed the maximum value allowed by the field size. But this assumes that chances are nil that a message gets out of sequence by such a gap. If this cannot be ascertained, the number of fragments is limited to the maximum value for F-NR.

F-TOT (8 bits)

This is the total number of fragments created or expected at this level of protocol. It is intended for resource reservation.

E-level (4 bits)

This is the level of the protocol handling the unit of which this message is the last fragment. In the protocol tree, it is a pointer back to a higher level in a traversing process.

ALT (1 bit)

This is an alternation signal used for various purposes such as checkpoints, or conversational interactions. It corresponds to the P/F bit of HDLC procedures. It pertains to E-LEVEL, if any, otherwise to P-LEVEL, in the protocol hierarchy.
Table I—Fragmentation Scheme

<table>
<thead>
<tr>
<th>P-LEVEL</th>
<th>U-NR</th>
<th>F-NR</th>
<th>F-TOT</th>
<th>E-LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>a+1</td>
<td>c</td>
<td>1</td>
<td>f</td>
</tr>
<tr>
<td>3</td>
<td>a+1</td>
<td>c</td>
<td>2</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td>a+1</td>
<td>c</td>
<td>3</td>
<td>f</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>a+1</td>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

COM (4 bits)

This is a command field allowing for a variety of interpretations of message formats.

ACK-ALL (16 bits)

This is piggy-back information intended to carry acknowledgment and allocation of messages for the reverse traffic. No assumption is made in this paper about the particular scheme used. The ACK-ALL field may be optional. It can contain a level number so that it be forwarded to the proper level of protocol, independently of the P-LEVEL of this message.

Coding and packing these fields toward an efficient format is left for further study.

Principles of operation

A simple example is likely the best way to explain out how transfer mechanisms work out.

Let us assume that a particular level of protocol has to fragment and transfer a unit of information with the parameters indicated in (Table I), row 1. The number of fragments generated will be f. Thus, it produces a series of messages with headers indicated in rows 2-6. The last fragment identified by the presence of E-LEVEL carries the parameters intended for the protocol at level e, which is to reconstitute a complete unit. Thus the receiver part of the protocol at level a is able to piece together all the parameters of its message, check it for validity, and make up a unit for its next higher level. If there is no E-LEVEL indicator in row 1, then the last message in row 6 will carry the value a for E-LEVEL.

There is no need for separate initialization, except at the highest level. Error and flow control may be exercised independently at any level on an optional basis. It is possible from the highest level to turn these options on and off, through appropriate commands directed to the proper level. A negotiation protocol may be introduced, so that the implementation of options be also optional. Due to these facilities, the ultimate user is in a position to select an efficient set of characteristics throughout all levels.

INTER-NETWORKING

The inter-connection of separate networks raises a number of problems which have been addressed in previous papers. One particular issue revolves around the maximum message size accepted by each network. A first approach is to assume a minimum size which every network would be able to carry. Thus inter-network traffic would use a maximum message size commonly agreed as a standard. This is typically what has been defined as the "datagram" service within the CCITT. A comprehensive inter-connection scheme based on this approach can be found in Reference 8.

Another approach assumes that some networks will offer larger message sizes than others, and that inter-network traffic should take advantage of larger sizes as much as possible. To that effect networks would be interconnected via gateways which would fragment messages further whenever they happen to be too large for the network downstream. A controversial implication of that scheme is that a network accepting only small messages would induce additional overhead in all successively traversed networks including the final receiver host. Also the fragmentation scheme in gateways would have to match tightly the E-E protocol between hosts.

It does not appear possible to do away with these drawbacks, which are intrinsic to any fragmentation scheme located within communication networks. To the extent that the implications of inter-network fragmentation are acceptable, then the protocol presented in this paper applies perfectly to the situation.

Indeed, no particular assumption has been made about the mechanisms transferring messages across protocol level boundaries. As a possibility, successive levels of sender protocols may be located in the gateways of several interconnected networks. Whenever necessary a message passing through a gateway will be broken down into several fragments. At the final destination, all matching levels of receiver protocols coalesce into a single procedure putting back together all arriving fragments through a recursive application of the protocol at the level indicated in each message header.

If at all useful, error recovery and flow control can be exercised between the final destination and any of the sender levels, including the traversed gateways. This may
improve efficiency by reducing repetitions, and smoothing out traffic.

Furthermore, if it turns out that for a certain destination all messages created at a certain level of fragmentation must actually travel through a single gateway, it may well reassemble pieces into higher level-units, rather than let them travel individually. This would reduce overhead downstream. Such a case is typical of networks requiring an initial virtual call set up. Then all fragments are constrained to leave the network through the same gateway. This is indeed a rare case where the clumsiness of virtual calls is put to an advantage.

STANDARDIZATION

There is not yet a final consensus for an HDLC standard within ISO. However proposals tend to stabilize around an independent double numbering scheme, with provision for address and control field extensions. As such HDLC would not meet the requirements brought up in this paper. Nevertheless the principles introduced by HDLC are reasonably close, viz:

- independent double numbering
- time delay between sending and acknowledgment
- data transparency

Field extensions and additional modes of operation could turn HDLC into an acceptable multi-level transfer protocol such as described here. Furthermore, a simple point-to-point physical data link is just a degenerated network, and could be under control of a network protocol. Thus at some point in the future, a general network protocol, with negotiable options, might be suitable for most kinds of information transfer, regardless of data structure, message size, and communication medium.

The recursive approach, combined with implementation options, should lead to simple and compact micro-programmed protocols integrated within intelligent terminals. On the other hand, if levels of protocols are distributed within a system hierarchy, each level may be implemented with a particular optimization in mind, while the complete structure remains consistent and easily understood.

REFERENCES

4. Denjean, F., Connexion de Terminals à un Rèseau de COMMUNICATION de Paquets, Internat. meeting on mini-computers and data communications, Liège, January 1975, 8 p.