Discussed are three major classes of data descriptor messages—for language processors, for input/output devices, and for resource allocation—required by computer networks where the details depend on the variety of systems in a network and on file complexity.

Three classes of networks are also discussed—remote job entry networks with compatible operating systems and computers, networks for transmitting arbitrary data sets between systems that have the same internal data representation but different hardware and operating systems, and networks for transmitting arbitrary data sets between arbitrary systems. The first two classes are illustrated by IBM *networks and the third by an interuniversity network.*

Requirements for the three classes of networks are compared. Further details of networks of systems with the same internal structures but different hardware and operating systems are given.

Describing data in computer networks

by D. H. Fredericksen

Programmers are quite accustomed to describing data they use: where to find it, what devices are involved, the layout of the data, what to do with it after the job is complete, processing techniques employed, and often a great deal more. The reason data is so described is that, nowadays, programs call upon a great many other programs for assistance. All sorts of commonplace functions that the programmer takes for granted are provided by other, already-written programs, that are invoked explicitly or implicitly by what the programmer himself writes. Thus programmers must describe things about their data to be able to use the functions provided.

In new system configurations discussed in this paper, where computers are linked together into general-purpose networks, the user avails himself of services on more than one system. Now he has to meet the data description needs of all the programs he invokes on all the systems he uses. For example, whenever he moves a collection of data from one system to another, he must describe that data sufficiently for retrieval at one system and reception at the other. We are speaking here of linkages among more-or-less independent systems, each with its own mission, but with a certain capacity for swapping services

with other systems. Network linkages, in current practice, are usually telephone lines or private wires. (Higher-capacity links are being used in the ARPA Network and in military networks.) Physical remoteness between the systems is possible, but is not a necessary part of the concept; there may be sound reasons to link computers within the same building.

The essential nature of the kinds of networks discussed in this paper is that they attempt to make the services of a variety of systems available, in whole or in part, to the general user of each system—without, however, tying the whole complex into a tightly knit super-system. They are not oriented toward any specialized application, but aim to pool facilities for all purposes that the needs of programmers and ingenuity of systems engineers may suggest.

The participating systems in such a network may be the same, but more commonly they are a rather diverse assortment. Access to "more of the same" might motivate the pooling of systems through a communication network if the component systems are all identical. The incentive is even stronger when each system has a specialty or two that the others lack. For example, one might be a paging system, especially advantageous for operations such as text editing and on-line debugging; another might be a batch system with a massive main storage that is more economical for production runs. Linked into a network, the two example system types offer the combined clientele the advantages of both types.

Thus, the differentness of the systems participating in a network is one of the incentives for its existence. At the same time, it is a source of complications—just as being more and more things to more and more people introduces complexity into an operating system.

Data description in a single system

Before discussing complications that a computer network environment introduces into data description, it might be well to review some principal forms of data description required of the user of a single system. These forms are grouped by the types of system programs to which they are addressed.

Aspects of the data must be described to such a language processor as an assembler, compiler, or interpreter. A language processor is responsible for assigning locations to data and instructions, and for converting symbolic references into their numeric form for use by the computer. A language processor is also responsible for generating sequences of machine instruc-

tions that carry out the intent of macroinstructions or executable statements. For these purposes, the language processor needs to know such things as the following:

- Individual constants and variables to be used.
- Characteristics of each variable with implications as to storage length, alignment on word boundaries, instructions to manipulate it, numeric or alphabetic, fixed point or floating point, decimal point insertion, etc.
- Larger groupings of variables e.g., arrays, tables, fields, records, lists.
- Relationships among different variables and structures, e.g., overlapping arrays in FORTRAN; fields of a work area in COBOL that correspond to the different fields of an input record.

There are also things that one must describe about data for the system input/output (I/O) routines, which perform all I/O operations on the system. This may further require overlapping I/O operations with other processing and with each other to the maximum extent possible; coping with a wide range of I/O devices; and dealing with a variety of file processing conventions. Therefore, I/O routines are given such information as the following:

- Maximum size of physical records for establishing buffer size.
- Number of maximum-size buffers and possible sharing among different files.
- Blocking of logical records within physical records.
- Record sizes constant or variable.
- Self-descriptive, variable-length record.
- I/O devices required.
- Labeling of tape files.

As a final group of data attributes, consider those required for the system resource management routines. These programs locate data that already exists, or find a place to store new data. Resource management routines schedule the use of the data, not only to smooth the transition between jobs, but also to permit concurrent use of the system by a number of users. They also perform certain forms of postprocessing on the data, and such further handling as may be involved in job breakdown. In some operating systems, these routines enforce ownership and privacy rights in various collections of data, and control the configuration of units on which the data is processed. Typical of the information acquired by resource management routines is the following:

- Name of the data set.
- Data set exists or must be created.

- Device type on which data reside.
- Reel or pack of data residence.
- Tape file or disk track of data set residence.
- Space required for new data set.
- Reel or pack mounting required.
- Data set readability by user and other programs at same time.
- Postprocessing requirements.
- Disposal of data set and storage after postprocessing.

The network environment

Consider now some progressively more ambitious types of linkage among computers, working up to a network in which arbitrary collections of data may be moved about among the member systems. In connection with this last stage, a full-scale problem of data definition arises. To simplify the discussion, we generally consider the situation between just two systems, mentioning third systems only when their presence affects a principle. We speak of a remote—or target—system and the user's own system—or home system—as a matter of arbitrary perspective, adopted for convenience in distinguishing two systems. The systems may be physically close, and the user may have access as readily to one as to the other. The user's *own system* is the one from which he initiates whatever transaction we are discussing.

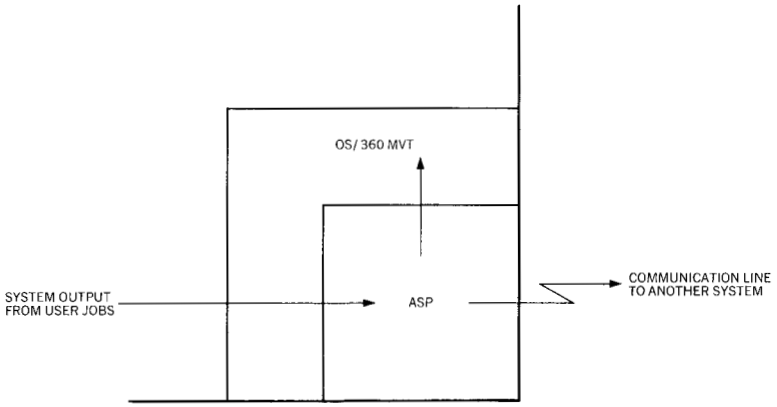
remote job entry network

If a user wants to submit a job to a remote system, then the operating system that schedules jobs at the remote system must be listening to a communication line as well as to its local sources of input (card readers, etc.). The job image that the user sends must include whatever job control cards are usual at the remote system.

That much is fairly obvious, but note that if the user's own system is also a batch operating system, he is now involved in writing job control cards for two jobs: one for his own system to transmit the remote job, and the other for the remote job itself. Typically, that does not end the matter. When the job is completed at the remote system, the user wishes to receive back his output. Remote job entry (RJE) is the term for this procedure.[1,2] For example, the remote system may merely assemble a user program for later execution by the user's own system. In this case, it is preferable to receive punched-card output images over communication lines instead of by mail.

Such an action brings still more system programs into play, both remotely and locally. At the remote system, a program that normally controls the punching must redirect its efforts to send the card images over a communication line. And some user program(s) must stand ready to receive the card images and make them accessible to the user.

Figure 1 Relationship of System/360 to the Attached Support Processor



In practice, the user may not want all of his output transmitted to his home system; or he may want copies at both systems. He may even wish to direct copies to third systems. In short, he needs a means to indicate his intentions—and this, as always, implies further control cards, or further parameters on existing ones.

Remote job entry, such as we have been describing, is today a fairly commonplace facility. For example, the Attached Support Processor (ASP) system,[3] used with the MVT level of Operating System/360 (OS/360 with multiprogramming with a variable number of tasks), offers such a facility. Briefly, modules of ASP collect job inputs from various sources, perform some scheduling functions over and above those of OS/360, and then feed the jobs to the OS/360 system tasks that normally handle such job inputs. Similarly, ASP modules intercept any output that the user designates as a system output to be printed or punched by system routines. Other ASP routines can then either put out the data locally, or send one or more copies to other systems via communication lines.

This situation is visualized in Figure 1. ASP (considered here only as a communication interface) is interposed between the regular operating system and the communication line(s). It gives remote systems a pathway to the local job input stream, and it gives the local job output stream a pathway to outside systems.

interactive-batch network

That takes care of entry and egress at the remote system, but what about the situation at the user's own system: how does he get through to the outside world; what program stands ready to receive the returning output of his remote jobs? A small but useful network that is operating at the IBM T. J. Watson Research Center may prove illuminating in this regard. An evaluation of this interactive-batch network, is given by Hobgood.[4]

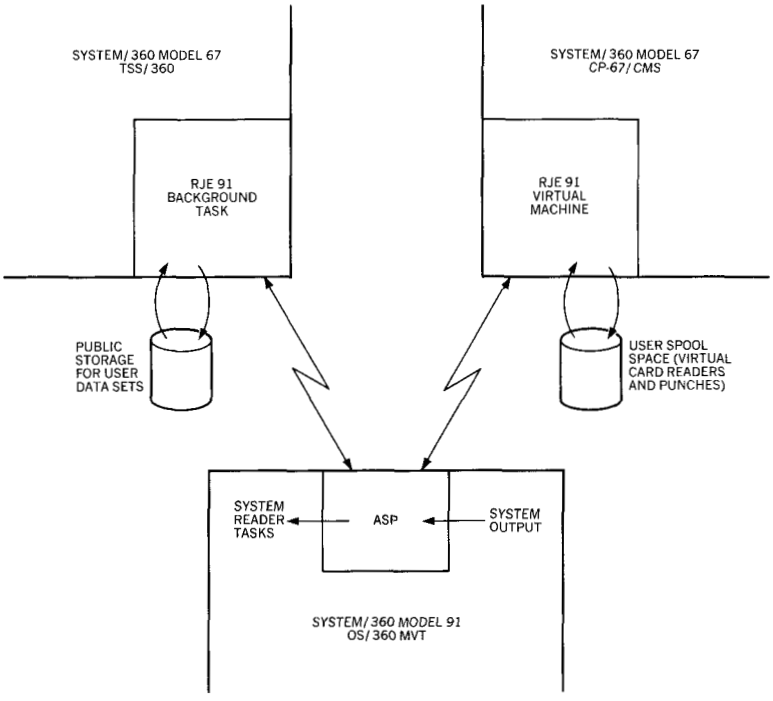
The network includes three models of IBM System/360: Model 91, running under OS/360 with ASP (in the local mode); a Model 67 that runs under the System/360 Time-Sharing System (TSS/360); and another Model 67 that is run under Control Program-67/Cambridge Monitor System (CP-67/CMS).[5] Each system has its own advantages, and there is every incentive to parcel out the stages of a job to the systems that do them best. For example, a particularly appealing operation is to edit program source decks at one of the time-shared systems (either TSS/360 or CP-67/CMS), and then ship the edited decks for assembly to the Model 91 (which does that function more efficiently). The resulting object decks can be run on the Model 91, or, in the case of CP-67/CMS, returned to the time-shared system for on-line debugging runs.

The ASP system at the Model 91, however, does not stand ready to accept input from just anywhere. A communication line must be available, and ASP must be notified by the system operator to start a reader program to read from that line. This is scarcely a convenient operation to repeat once for each remote job that is entered into the Model 91. Neither can each user afford to keep a private wire open all day for the jobs he might submit. The solution is to have a communication interface at each time-shared system that provides communication services comparable to those of ASP.

Such a service is provided by a program called RJE91 that operates continuously at each of the time-shared systems. Though the basic functions remain the same, the implementations of RJE91 differ at the two systems, to conform to the different structures of TSS/360 and CP-67/CMS. RJE91 serves as a funnel from the users into the communication lines, and from the outside world back to the users. Source "cards" can thus be sent to the Model 91, and object "decks" returned without actual punching being performed. Prospective printouts can also be sent back, and can be edited at the time-shared system before or after printing.

Figure 2 shows in schematic form the remote job entry network. At each system there is a communication interface—ASP at the Model 91, RJE91 at the TSS/360 and CP-67/CMS systems. The network is not symmetric, however, between the Model 91 on the one hand and the time-shared systems on the other. The interface at the Model 91 connects the outside world with the system's job input and job output streams. The TSS/360 and CP-67/CMS interfaces connect the outside world with the public storage and the spool space of the individual users, all of whom may use the interface. Which is to say, while traffic flows in both directions, the user at the time-shared system can contact the batch system only for purposes of submitting jobs. A job run-

Figure 2 Remote job entry network



ning at the Model 91 can contact one of the time-shared systems only to forward a job output to a user after the job has completed its processing. No contact is possible between a user and his remote job while it is running, nor between user programs running simultaneously on two systems.

more generalized communication interface

The principal motivation for the remote job entry network we have just been considering is that it takes advantage of the differences among three operating systems. By this criterion, the limitations are harmless, because there is normally no advantage in moving a job from a batch system to a time-shared system. Also a batch system user would not wish a job in the batch system to tarry while that system interacts with a remote user.

There are other promising network applications, however, for which such limitations are more serious. Suppose, for example, that the principal attraction at another system is not its operating system, but a program that has to be run there. This might occur when the program uses a device that is not available at the home system. The program might have to be used remotely because it is proprietary, or perhaps because it would be too costly to duplicate on the user's own system. Thus motivated, one may seek a more comprehensive facility than remote job entry. The situations that follow give insight into some desirable capabilities of a more generalized remote job entry facility.

Assume that one wishes to use a remote program as a kind of remote subroutine of a program running on his own system. That is, one might initiate a job on the remote system, then communicate input values from a program running on the home system and receive output values from the remote program. Needed for such an operating mode is a communication interface that allows user programs to interact with each other as well as with the remote program.

A further requirement may be to forward more arbitrary forms of input than those that can be embedded in the job input stream. The remote program, for example, may require data saved from a previous run, and may expect the data from tape or disk. The user, however, does not necessarily have permanent storage rights at the remote system. He needs to transmit the data ahead of his job and put it temporarily on the required medium. Likewise, one may have to retrieve more arbitrary forms of output than those declared as system output. System output is normally stored temporarily on spool space. Limitations on the system output format are dictated by the system routines available to postprocess it. Again, having worked up his data at a number of systems, a given user may then have to gather it together for a new job at any one of them.

Load leveling and system backup are two further network applications that require the transmission of arbitrary data sets from system to system. In load leveling, a job and data sets used by the job are shunted from a busy system to one whose current processing load is lighter. The same is true when jobs are sent to remote systems because the home system is down, assuming a functional auxiliary home system and a communication line.

To handle conditions such as these, a communication interface was designed and implemented at the IBM Research Center that offers two new services. (1) Access is possible from a user program on one system to a user program on another. Programs that produce output other than system output can then, in many cases, redirect the output to remote systems. (2) Data sets are transmitted by the communication interface, or by programs that the interface invokes by itself, without further intervention by the user. The communication interface operates with the MVT level of OS/360, at which level a number of user jobs may be simultaneously working their way toward completion. At the same time, system tasks accumulate input for other jobs and postprocess the output of completed jobs.

In this milieu, the Communications Interface Task (CITASK) operates as an additional system task that funnels messages from the user jobs into the communication lines, and from the communication lines to user jobs and/or the system input tasks.

To either user jobs or system tasks, CITASK looks like a magnetic tape unit. Thus writing to certain dummy tape devices actually forwards material to CITASK, and reading from such devices requests inbound messages from CITASK. The function of CITASK is discussed in more detail in Reference 6. Here we simply summarize CITASK by observing that this communication interface or network subsystem, of which CITASK is a central part, sends and receives arbitrary data sets. Given a description of a data set, CITASK arranges for other components of the network subsystem to perform the requested operation of either sending or receiving a specified data set.

Because CITASK puts jobs into the system job stream, and because these jobs, like any user jobs, can communicate through CITASK, the following two ways of forwarding data sets are available. (1) CITASK generates a support job, which invokes a utility program provided with the network subsystem. This utility reads the data set from its storage device on the local system and writes it into the remote system via CITASK or vice-versa depending upon whether the request is to transmit or receive a data set. (2) CITASK attaches a subtask to do the same thing.

Describing data for the network

scope

It is necessary that CITASK be given a description of the data set adequate for retrieval at the sending side or for placement at the receiving side. Earlier in this paper we discussed various forms of data description that are required by a single system. That discussion is now enlarged upon for describing data for network usage.

We previously noted three major classes of data description:

- Details about variables and data structures required by a language processor and possibly by an application program
- Information used by system I/O routines for transferring data between external media and main storage
- Data descriptions needed by system routines that allocate resources

The network subsystem just discussed does not concern itself with data descriptions required by a language processor. Rather, the network subsystem moves arbitrary bodies of data from external storage on one system to external storage on another (or into the job stream of the target system, if that is appropriate). Optionally, the data can be printed or punched at the receiving system. Some reorganization is possible, but only that which affects I/O routines at the target system, e.g., changes in record format or blocking factors. It suffices simply that the data be

describable to the system I/O routines and resource allocating routines. Further analysis of the data into fields, variable types, arrays, and so forth is deemed to be the user's responsibility.

Most batch operating systems allow data attributes in the second category—those required for the input/output routines—to be specified either in the user's program or in control cards that accompany his job. Resource-allocating attributes, on the other hand, are typically communicated only by control cards. Thus, the combined information may perhaps not be thought of as a single data description, and one may reasonably quibble whether an item such as device type really describes data. However, relative to the network subsystem's use of the data, the one kind of information is as necessary as the other. As a matter of fact, the network subsystem carries this perspective even farther. For example, accounting information, which is required to authorize the service of forwarding or receiving the data, is also part of the data description. If such information were not provided as part of the data description, it would have to be provided separately. It is convenient to bring all the required information under one roof.

protocol

This is not to say that every one of the attributes listed under one category or another must be stipulated explicitly on every occasion by the user. Much of this information may be available at the sending side in a system catalog, in labels associated with existent data sets, or by the application of default values. For this reason, the support program that is invoked by CITASK at the sending side is responsible to embroider the data description it has been given by the user, and forward it to the target system in a (generated) request to receive the data set.

The step-by-step interplay proceeds as follows. The user writes a protocol message to CITASK at the system (not necessarily his own) that is to send the data set. This message, in its simplest form, contains only sufficient description to authorize the service and locate the data set. A discussion of a more comprehensive network protocol is given by McKay and Karp in this issue.[7] In response, CITASK invokes the proper support program for sending the kind of data set in question. The support program composes a message for the target system, and requests that the data set be received. All information that the support program finds out about the data set—from labels or otherwise—is merged with the description provided by the user. Thus enlarged, the description is received by CITASK at the target system, and used to invoke a suitable program for receiving the data set.

The user can disengage from the process as soon as the initial protocol message has been acknowledged by that system. Also,

third-party requests are possible. That is, a user at system A can request system B to send a data set to system C.

Thus far we have described a network in which CITASK and OS/360 MVT are installed at both the source and the target system. The network subsystem, however, was designed so as to be easy for other types of systems to link with it, as long as they match the subsystem's line discipline and cooperate with the sort of interplay sketched in previously. It is generally undesirable to limit a network to systems of one kind, for, as we have found, the very differences among systems can be a powerful motivation for linking them together.

Having accepted this objective, there are two major complications in the description of data. Different systems use different representations of data and different file conventions, as well as different external storage media, resource allocating procedures, etc. Data descriptions that are meaningful at the source system may have to be specified in different terms for the receiver. Also, even where the two systems are alike, the data description at the source may have to differ from its description at the target system. This is particularly true of where-to-find-it information, but other attributes frequently differ as well.

The first of these difficulties becomes particularly acute when data is moved between systems with radically different system architecture. Suppose, for example, that one system represents integers in a 32-bit word, whereas another uses words of 36 bits. Perhaps one system represents characters for external display in an eight-bit code, and the other uses six bits. One system may devote seven bits to the exponent in floating-point numbers, and the other devotes only six. When the differences between the systems are this severe, the simple transfer of records from external storage at the source to external storage at the target is inadequate.

To compensate for such architectural differences, either of two procedures are used. Each application program that is to use the records must be coded with a specific knowledge of the conversions to be performed upon the specific records that are sent to it. Alternatively, each data set must be preceded by a description not only of its file characteristics sufficient for placement on external storage, but also of its contents, down to the level of individual variables and their decomposition into bit strings. This description must itself be in a form that can be interpreted anywhere in the network.

The first alternative would be an impossible impediment to network usage. It would require every application program that used data from afar to be written with that fact in mind, and with

a specific donor in mind. Only the most highly used applications would be worth the effort. The hope of sharing programs that are written at remote systems is frustrated under the first requirement. The necessity of doing things that way might serve to characterize a special-purpose network as opposed to a general-purpose one.

This difficulty has been widely recognized, and there has been a great deal of interest in working out a standard data description language. A description in such a language would accompany each data set transmitted from one system to another and be used in either of two techniques. Each application program that is intended to be shared through the network is designed to look for the data description associated with each data set it uses, and make any necessary conversions. Alternatively, utility programs are available at each system that uses the data description to convert the data set into a standard local representation.

the MICIS standard

One design that opts for the first of these approaches has been put forth by a group of users associated with the MERIT computer network.[8] This network connects the three largest universities in Michigan. Two of the three systems in the network are IBM System/360 Model 67's; the third is a CDC 6500. Various general-purpose statistical application programs are available at the different universities. A proposed Michigan Interuniversity Committee on Information Systems (MICIS) standard for data description aims to facilitate the transmission of data among the systems for end use by these programs.[9]

The data description itself is written in a standard set of characters that are encoded the same way in all computers of the network. Each data description starts with a bootstrap record that indicates the layout and placement of the data description itself. (Parts of the description may be interspersed with groups of data in the file.)

If necessary, the bootstrap then describes the representation of various types of numbers present in the file (reflecting machine-dependent formats). This is followed by an indication whether further descriptions are in terms of rows or columns. (Each actual group of data is transmitted as an array.) Then there are descriptors for the location of each variable within the typical (data) record for the group, its format (possibly one of those for which conversion requirements were specified earlier), etc.

In short, data is described down to the level of individual variables, grouped in arrays. This is the kind of data description discussed earlier under the scope of the required description in the first category: descriptors required by language processors and/or application programs. There is no escaping this level of

description if radically different systems are to cooperate usefully in the network.

While the approach is a necessary one for networks that include heterogeneous computer types, it presents some obvious difficulties. First, application programs must be written to take account of the information in the data description. Programs cannot simply be written for local use and then shared as an afterthought. They must be written as required, and then they can accept data from anywhere. Second, it is a formidable (and perhaps impossible) task to invent a single set of data descriptors to cover all the known applications in a manner acceptable to everyone who uses it.

The MICIS standard is avowedly oriented toward statistical applications, for which reason it includes some unusual descriptors, such as the specification of missing values. To adapt MICIS equally well to other applications, and to the way of thinking of non-FORTRAN programmers, would probably require an excessive amount of additional detail. Indeed, no limit is in view. Each new class of applications that arises is likely to require special-purpose descriptors analogous to the missing-value descriptors of statistical usage.

The authors of the MICIS standard liken their data description to user labels on a tape. The comparison is apt, since making use of the description is not a function of the network interface (i.e., of system programs), but of the end-use application programs. At the same time, the currency of the standard might be somewhat wider than one normally associates with user labels, since a broad class of potential users may be able to make use of the same standard.

network subsystem data description

The OS/360 MVT network subsystem, as we have already noted, offers a more limited service. It transfers data sets from external storage on one system to external storage on another. No effort is made to describe the data at the level of variables and bit strings for the benefit of end-use programs. This means that traffic between the network subsystem and any truly alien system is only minimally facilitated. When the data arrives at its target, the end-use program is on its own to decipher it.

While this is somewhat circumscribing, there are two weighty justifications for starting at this level. The problem of describing data at the level required for I/O and resource allocating routines is challenging in its own right, especially when the operating systems in the network include a wide variety of conventions and options. For the medium-term future, and perhaps even in the long run, the earliest and heaviest buildup of traffic is likely to

occur within families of closely related computers, not among conglomerations of widely differing systems.

Variety is a motivation for the linkage of systems into networks, but it is also an obstacle. Ideally, the kind of variety that is helpful can be achieved without the kind of variety that is most troublesome. It is attractive to contemplate the use of two operating systems whose strong points are complementary. The prospect of having access to other configurations of main storage and peripheral devices is similarly attractive. The use of another system under the same operating system, but with different options could also be very useful.

Coping with differences in basic data representation and instruction sets is difficult. In general, that kind of variety adds burdens rather than versatility. It is a cost to be weighed against whatever advantages may be offered by the good kinds of variety.

Among systems that use a common representation of data at the level of individual variables, the description at this level can be dispensed with. It is possible for such systems to share a program without its having been written with that objective in mind. It suffices to inform the remote user what data the program expects and how it should be laid out in the external records. No extra layer of description is required for data conversion.

When a sufficient set of descriptors has been determined for moving data between external storage on any two systems of the network, supplemental (and separate) descriptions can be worked out to deal with the further problems arising from heterogeneous data representations at the individual systems. The latter descriptions can be tailored to the requirements of different end-use programs, and can remain transparent to the data-moving routines. Thus their status is similar to user labels.

Input/output and resource allocating descriptions

We now discuss the more restricted problem to be handled by the network subsystem, that of sufficiently describing data to move it from the external storage of one system to the external storage of another. The two systems may have different operating systems, but we assume that they have a common representation of data. Even these assumptions leave something of a challenge—different file conventions and different ways of conceiving and requesting resources, wherein such differences preclude a simple transfer of control card parameters and/or data set labels.

Even at a single system, the profusion of options and conventions offered by an operating system such as OS/360 MVT are impractical to use if the system does not offer a number of abbreviations describing the more common situations and does not offer ways of automating such descriptions by cataloged procedures, retrieval from data set labels, etc. Therefore, the language of data descriptions for network interchange should make full provision for abbreviations, summary descriptors, default values, and the like. Moreover, to the extent possible, such a language should be backed up at each system by software that supplies the information automatically.

double list

We noted earlier that even when the source system and the target system are of identical types, the data description that applies at the source system may have to be retouched for the receiving side. For example, the data set may reside on a private disk pack at the sending side and have a particular volume serial number. At the receiving side, however, the data may be placed on public storage assigned by the system. As another example, it may be desirable to shorten blocked records at the receiving side if the devices used there have a smaller track length.

For this reason, a double list is used as the basic form of the data description as follows

SND (. . . . . .), RCV (. . . . . .) or simply (. . . . . .), (. . . . . .)

where the (. . . . . .) represents contents, and SND and RCV are optional mnemonics that represent SEND and RECEIVE.

Within the parentheses are various parameters needed to move the data set from source to target. These parameters are attributes of the data, although, as previously explained, some have to do with the description of devices and other matters required for performing the service. Each attribute is considered as having two values—one value that applies to the sending side and another value that applies to the receiving side. For most parameters, these values are identical. Therefore, to save unnecessary writing, a convention is observed whereby the value of an attribute given in the SND list is assumed to apply to both sides unless it is explicitly overridden by exception values in the RCV list. If all values are identical at both sides, the RCV list is omitted entirely.

default values

The double list is a solution to the part of the send-receive problem wherein systems that use the same set of descriptors may require different values to describe the same data set. Also, both the source system and the target system may be remote from the system at which the transmission request originates. There is, however, a subtle point that might escape first notice.

Like most languages, this one endeavors to convey as much meaning as it can by unspoken agreement. Wherever the context requires that an attribute have a value, there is a default value. The user who wants to write programs in the language with any degree of proficiency must know the defaults. However, the default values that come naturally at the sending system differ from those at the receiving side.

For example, consider what to do with the data set if one of the two programs that cooperate in sending and receiving data sets should terminate with an error. At the sending side, in the absence of other instructions, one normally wishes to keep the data set, assuming it resided originally on external storage. At the receiving side, it is more natural to scrub the suspect copy that has been received, provided this can be done without disturbing its surroundings.

A consequence of assigning each attribute a value for the sender and a value for the receiver is that each value must also have two default values. A list that apparently calls for common treatment (default it at both sides) may mean to assign appropriately different values at the two systems.

data elements

A much greater difficulty is that different operating systems do not talk the same language about data, resources, or services. One attack upon this problem requires collecting every parameter that may be used by any of the systems expected in the network, analyze the parameters into their finest elements, and assign a name to each element.

A data description then consists of a list of values for these atomic attributes and is complete for the purposes at hand if both sender and receiver can get what they need out of it. The language itself consists, in principle, of a word for every need. Excess repertoire can always be ignored. In general, each system is expected to contribute all the information it can, and extract whatever information it needs.

The idea of analyzing the attributes into a common set of elements is that different systems can then reassemble the elements into the parameters that each system uses. Parameters tend to differ from system to system not so much in meaning as in the way each system bundles meanings. OS/360, for example, combines the following five different questions into the LABEL parameter of a DD card: (1) Which of several files is this one on a tape? (2) Are standard labels employed? (3) Is a password required? (4) Is the file going to be used this time for input or output? (5) How long should the file be retained after this use? Other information that bears upon the retention period is embedded in the DISP parameter.

Another operating system that handles tape drives is likely to require at least some of the information listed above, but there is no compelling reason why the other operating system should wrap particular values into one parameter, nor why possible values for the retention period should be separated into two parameters. Such groupings reflect only a particular implementation of the services that are called for.

Any given pair of systems could perhaps have programs for extracting what they needed out of each other's parameters, but that quickly gets cumbersome as more systems are encompassed and unusable if one system does not know the character of the other. It is better that each system know how to break down its own parameters into the common currency.

Of course, the immediate price is that each system must be able to extract from this common currency what it needs, and assemble the particles into its own-style parameters. CITASK and its support programs do exactly this and ignore or pass along any information they have no use for.

Needless to say, this approach offers no way to describe at one system another system's facility for which it has no functional equivalent. Under CP-67/CMS, for example, a CMS virtual machine offers no facilities for processing an Indexed Sequential data set. No combination of elementary attributes can be expected to describe such a data set to CMS, when no such data organization is within its ken. But then, no such description is needed at a system that can do nothing with it.

On the other hand, comparable facilities that have different names at two systems can be described straight-forwardly, using the two-valued attribute feature. For example, under appropriate conditions, a Physical Sequential data set from a system under OS/360 can be sent to a system using TSS/360 and filed as a Virtual Sequential data set. An appropriate description of the data is as follows:

SND (. . ., DSORG=PS, . . .), RCV (. . ., DSORG=VS, . . .)

This does not imply that an interface comparable to CITASK, that is, one capable of acting upon this description, has been implemented for TSS/360. Here we are talking only about the adequacy of the language.

some specific parameters

All possible contents of the SND and RCV lists are not detailed here.[6] Some highlights, however, that show interesting touches due to the network environment are presented.

Inside the parentheses, both lists may consist entirely of keyword parameters separated by commas or they may start with a

single positional parameter optionally followed by keyword parameters. To keep sublists to a minimum, an effort has been made to define the attributes so that each operand is atomic, that is, operands do not involve multiple subparameters.

First, we take up the positional parameter, which optionally heads the list. Writing any value for this parameter is interchangeable with writing the bundle of keyword parameters stated to be its equivalent. Some keyword parameters are not detailed here, but the context should make their meaning clear.

positional parameters

Bundled attributes are provided for ease of use, whereby standard bundles of attributes are defined and given network-wide names. One or another of these names may be used as the positional parameter of an attribute list. If so used, it absolves the user of writing any keyword parameters that are encompassed in its meaning, unless he wants to override part of the bundle he has invoked.

As always, the desire to say things in few words competes with the need to have words for everything. Having broken down the attributes into elementary components so that each system in the network can recombine them in its own way, we apply names to the following recombinations that are common for the purposes of the network.

CARDS describes card images that are stored and transmitted in fixed blocks, which is tantamount to writing RECFM=FB, LRECL=80; nothing more is implied.

JOB describes card images that are stored and transmitted in fixed blocks. At the receiving system, however, the card images are to be fed into the job stream. Specifying this attribute is equivalent to writing RECFM=FB, LRECL=80, and QUEUE=RDR.

TAPERECS is the first (data) file on one reel of a tape that conforms to local standards on each system. It is considered legitimate for standards with respect to header and trailer labels, density, and device type to differ at the sending and receiving systems. The communication interface is expected to make its own provisions for describing the type of device implied at its own shop.

Header and trailer labels are not transmitted as such. If, however, header labels are standard at the sending side, the sending program is expected to expand the user's description from the information in the label so that labels may be constructed by the receiving system if needed. Of course, if header labels are not standard at the sending system, the user must provide in his

description any information necessary to the receiver that is ordinarily gleaned from the headers.

There is no precise keyword equivalent for TAPERECS, which is equivalent to writing UNIT = value1 and LABLTYPE = value2, where value1 and value2 are standards in force at the system where TAPERECS is interpreted.

This parameter is especially useful in a network because the user may know that he wants his data put on tape, but may be unfamiliar with the labeling conventions and unit types at the remote system. There is an analogous keyword for disk records.

keyword parameters

We turn now to the keyword parameters. Unlike the values just described for the positional parameter, keyword parameters need not be meaningful throughout the network. The subset chosen applies to the systems involved and to the case at hand. The value given to an item by writing a keyword parameter (other than DESC) takes precedence over any value it may be given by an attribute bundle called for in the same list. The effect is as though each bundle were a cataloged description made up entirely of substitutable parameters.

Prearranged data set descriptions that make sense throughout a computer network are difficult to define, because of the variety of operating systems that may be encountered. At the same time, much of the traffic may take place within families of systems, the members of which could easily agree among themselves as to what is useful and common. Also, a handful of descriptions may cover ninety percent of the information required in ninety percent of the cases at a particular installation.

To provide for descriptions whose currency is only local or informal, the following keyword parameter is provided:

DESC = description-name.

This parameter, like the positional parameter of the previous discussion, names a bundle of attributes. The difference is that the positional parameter names a bundle that every party to the communication network undertakes to interpret according to standard specifications (insofar as possible), whereas DESC names a description that is prearranged among some group of users and the installations with which they deal. In the latter instance, it is the user's business to know what the description means at the system(s) he addresses.

The DESC parameter can be thought of as invoking a cataloged description although nothing is implied as to the location of the catalog (i.e., in external storage or internal to the communication interface).

Editing procedures may be specified by users to reorganize or edit data in ways that are not provided for by the communication interface. Editing could in principle be accomplished by a separate job, but it is often more economically performed in the process of sending and receiving the data. Both to accommodate the user and to relieve the interface of providing for every possible case, therefore, the following escape hatch is provided:

EDITOR = procedure-name or E = procedure-name

specifies a procedure or program that does the sending and/or receiving (depending which list(s) it occurs in).

Processing queue is provided at the receiving system so that the user can direct the data set to be placed in a named processing queue. The form for this request is as follows:

QUEUE = queue-i.d. or Q = queue-i.d.

Also there is the following standard form:

QUEUE = RDR

which has the meaning that the data set is to be placed in the (batch) job stream of the system. This feature illustrates an interesting point of view that arises in a network environment. From the standpoint of the communication interface, passing blocks of data from the communication line to a support job that stores them away on external storage is pretty much the same operation as passing blocks of data to the sysem program that takes in jobs. Hence, to submit a remote job, it suffices to ask the system to receive a data set with the special attribute Q = RDR.

At an OS/360 system, if queue-i.d. is a single character in the range A-Z or 0-9, it is taken as a queue being processed for output, that is, in OS/360 terminology, an output class to be specified in a SYSOUT parameter.

User identification has the following keyword:

USERID = name-of-user or U = name-of-user

This information provides accounting information for sending and receiving services not otherwise accounted for at an OS/360 system when a subtask rather than a support job is set up to provide the service.

User information is also used to log on at a time-sharing system such as TSS/360 or to compose a JOB card—when appropriate—at a batch system such as OS/360. The same information provides unique qualification for data set names when so requested.

Data set name is a piece of information that is generally needed when a remote system is asked to send or to store a data set, and must conform to the standards of the system(s) at which it is to be used. For example, at an OS/360 system the data set name must not exceed 44 characters. The principal form of data set name is the following specification:

DSNAME = name or DSN = name

An additional keyword that is related to the naming of data sets is UNIQUE, which is dictated by the network environment, and is given as follows:

UNIQUE = YES-or-NO or UQ = YES-or-NO

When UNIQUE is YES, it asks the system(s) at which this value holds to prefix the data set name with some identifier that makes it unique. To wit, the symbolic ID of the user's system and his USERID are concatenated with the given name, so that the effective data-set name is

userid.unid.dsname.

Since the user may be in no position to check, this insures uniqueness of the data set name at a remote installation whose public storage may already contain a data set with the unqualified name.

The default value of this parameter is NO at the sending system and YES at the receiving system. Thus an existing data set at the sending side can be fetched by its ordinary name and renamed for the user's purposes at the receiving end without the user's having to say so. Subsequently, the user may need to keep the renaming convention in mind when referencing the data set in a job executed at the receiving system. He must also keep his data-set name short enough to accommodate the extra characters or else forego this protection by specifying UNIQUE = NO.

Direct access space requirements pertain when a data set is to reside on a direct access volume and space is not already reserved for it there. The receiving system must be forewarned of its size by the user, who writes the following information:

SIZE = nnnnnn or SZ = nnnnnn

where nnnnnn states the size in terms of blocks that are assumed to be of the length given in the BLKSIZE parameter that is effective for the RCV list.

If the user omits this information—and if it is available at the sending system—the communication interface or the support program to which it delegates the work should add available information to the user-supplied list. In the OS/360 MVT network subsystem, available information is supplied by the support programs. The necessity of the information depends on the context and on the conventions that are in force at the receiving system.

Note that the network environment constrains the size to be stated in terms of blocks because the direct access units at the receiving system may be of a different type than those at the sending system. That is, the capacities of a track or cylinder may not correspond, and size specifications in terms of these units that are valid at one side may be too large or too small at the other.

Volume access is another parameter that is influenced by the network environment. Accessibility relates to the physical volume of the disk pack on which a data set resides or is to be placed. This item of information is provided in OS/360 Job Control Language as a subparameter of the VOLUME parameter in a DD card. In that environment, volume access assumes either of the two values "public" or "private." In the network environment, it is convenient to make the item a separate attribute, and add the following additional value to its range:

VOLSTAT = status or VST = status

Here "status" may be given one of the following values:

PUBLIC (or PU)—public volume
PRIVATE (or PR)—private volume
NETWORK (or NW)—volume dedicated to network usage

PUBLIC and PRIVATE have their usual meanings. A NETWORK pack is a public volume that is dedicated to network usage in the following two respects: (1) it is permanently mounted whenever the communication interface is active; and (2) the communication interface is able to access it. In practical terms, this means that the communication interface itself can perform the task of retrieval or storage, thus bypassing the need to set up a support job.

The volume access parameter brings up the point that it may be desirable to distinguish between public status with respect to local users and public status with respect to network operations. Actually, since OS/360 lacks facilities to enforce the concept of ownership, the OS/360 MVT network subsystem as currently implemented does not do so either. Declaring a volume to have

NETWORK status merely makes it easier and more economical for the network user to have it transmitted from or received onto.

A number of other parameters are specified for commonplace attributes of the data and the external storage media on which it resides or is to be placed. For example, there are keywords for data set organization (DSORG), physical unit (UNIT), volume identifier (VOL), block size (BLKSIZE), record format (RECFM), and logical record length (LRECL). These and others are not detailed here because they have their usual meanings.

Concluding remarks

We have distinguished the following three major classes of data description that are typically supplied in a program or in control cards:

- Variables, fields, and other structures, together with their interrelationships as needed by language processors and application routines in a program library, etc.
- Input/output devices, file conventions, record layouts, etc., as they affect the transfer of data from external storage to main storage.
- Data set locations, size, access rights, disposition after processing, etc., as required by the system resource allocating routines.

Of these, some, all, or none may accompany a data set as a descriptor message whenever the data is transmitted within a computer network. It depends on the type of network—the variety of systems in the network, the variety of file types whose transmission is supported, and the degree of assistance to be undertaken by communication interfaces or other system programs.

In the IBM Research remote job entry network discussed, only minimal data descriptor messages are required. Those messages are the returning output from the batch system preceded by a record giving the user i.d., job name, and DDNAME with which the data set is associated in the job. The user is unaware of this record, since it is created by one communication interface and absorbed by the other. A minimal descriptor is possible in this example because of certain facts about the network design constraints that simplify data description. The batch system (OS/360) takes in only card images whose destination is known to be the system job queue. The system returns only output that can be handled by some ASP postprocessing module. The bulk of the work is done by the user, who describes his data to the batch system in the DD cards of the job itself. (An additional ASP con-

trol card is embedded in the job for each output data set that is to be transmitted.) The communication interface at TSS/360 or CP-67/CMS undertakes no more than to receive variable-length records, store them in a data set or a spool space associated with the user, identify them for the user by incorporating the job name and DDNAME into the data set name at TSS/360 or the filename-filetype at CP-67/CMS, and advise the user that the data set has arrived.

Discussed in another environment is the MERIT network, among three universities in Michigan that shows the impact of including different computer types in the network, particularly where they use basically different data representations and instruction sets. Since the user cannot send the application programs to a system with a different instruction set, it is attractive to send data to the program. But the ability to accept data from remote systems of unspecified type requires that the application or utility program be written to look for one or more associated descriptor records in a standard form that describes data down to the level of arrays, variables, and bit strings.

Also examined is another IBM Research network supported by an experimental OS/360 MVT network subsystem. Here the participating systems are assumed to be of like architecture, i.e., using the same internal representations of data. However, the hardware configurations, operating systems, and available services may all differ. It is desired to be able to call for the retrieval of an arbitrary data set at any system in the network, its transmission to any other system, and its storage, execution, or servicing at the target system on the basis of a standard descriptor message. At least some of the systems offer a challenging variety of options as to file conventions, resource specifications, and the like. With this for a problem, the need for data descriptions at the variable and bit-string level disappears. On the other hand, input/output and resource-allocating classes of descriptors become prominent.

As the description is provided for the benefit of system programs and/or utility programs at both the sending and receiving sides, the description must describe the data for both. This leads to double-valued attributes—and, in some cases, double-valued defaults, one for the sender, and one for the receiver.

One way to incorporate into a single description the information required by two systems that might offer different services, conceive of resources differently, and talk differently about things like files is to analyze the parameters used by each system into atomic elements, with as much commonality as possible. The description is then encoded in terms of these elements and put together again at each system in its own way. As much as possi-

ble, the onus of this description is placed on the system programs and utility programs themselves. The sending utility program, for example, rounds up as much information as it can from data set labels and other sources and adds this to the description provided for the receiver.

The data description language used by the OS/360 MVT network subsystem copes with these problems. It includes in each description a SEND list of attributes and (to the extent that they differ) a RECEIVE list. Defaults are systematically worked out to be natural for the side(s) at which they apply. The breakdown of attributes into atomic components is compensated by provision for bundled descriptions that put them together again in such groups as may prove convenient.

The network influences certain commonplace parameters, such as data set names, which may need to be rendered unique by a convention observed throughout the network.

For the near-term future, it is likely that most computer network traffic will be of the RJE variety among computers of like type. For this, elaborate communication protocol and data-description messages are unneeded. Protocol messages that pass between systems can be generated by system programs. The user's role consists merely of supplying control cards for his job, with perhaps some extra cards to control routing of the output. If the user wants to get back data other than the usual printed or punched output, he must do something in his program to put the data into one of those two forms and then reconvert it at his home system.

Even in the long run, it is unlikely that massive data sets will be transmitted across communication lines. There are economic limitations and time constraints on this. It is foreseeable, however, that as network usage builds up, there will be a mounting demand for services that shuttle moderate size data sets of all types among systems with a minimum of rearrangement by the user. This prospect prompts interest in network-oriented data description languages such as have been discussed. Naturally, users want to write the least they can in any particular context and to have the ability to express anything that needs to be expressed on occasion. Reconciling these objectives is a key step in making computer networks attractive.

ACKNOWLEDGMENTS

The author wishes to cite J. W. Meyer and R. Nachbar for their roles in the development of the remote job entry system (RJE91) for the System/360 Model 91.

CITED REFERENCES

1. D. W. Barron, *Computer Operating Systems*, 23 and 105–106, Chapman and Hall, Ltd., London, England.
2. The Comtre Corporation, A. P. Sayers, Editor, *Operating Systems Survey*, 79–80, Auerbach Publishers, New York, New York (1971).
3. *IBM System/360 and System/370 Attached Support Processor System (ASP), Version 2, System Programmer's Manual*, Form GH20-0323-8, Ninth Edition, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604 (March 1971).
4. W. S. Hobgood, "Evaluation of an interactive-batch system network," *IBM Systems Journal* **11**, No. 1, 2–15 (1972).
5. R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal* **9**, No. 3, 199–218 (1970).
6. D. Fredericksen, L. Loveless, J. Rooney, R. Ryniker, S. Seroussi, and A. Weis, *OS/360 Network Interface User's Guide*, IBM Research Report RA 23 (#15638), IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 (1971).
7. D. B. McKay and D. P. Karp, "Protocol for a computer network," in this issue.
8. *The MERIT Computer Network: Progress Report for the Period July 1969–March 1971*, Publication 0571-PR-4, Reproduced by the National Technical Information Service, PB 200 674.
9. M. Donaldson, S. Robinovitz, and B. Wolfe, *Preliminary Draft: Proposed MICIS Standard for Data Description*, Michigan Interuniversity Committee on Information Systems (December 1970).