Data communications: The implications of communication systems for protocol design

by B. C. Goldstein
J. M. Jaffe

The construction of a communication network architecture, specifying protocols by which systems communicate, is a complex art. Much has been written about the optimal protocols for theoretical models of systems. This paper points out that protocol design must depend on the "nuts and bolts" of the systems which implement the protocols. Numerous examples are provided to support this thesis. The paper also briefly discusses other issues that influence protocol design and draws lessons for standards activities.

In the past several years there have been a significant number of publications describing various aspects of computer network protocols.[1-5] Papers have described specific network architectures,[6,7] standards activities,[8] particular protocols,[9] performance of protocols,[10] local-area networks (LANs), wide-area networks (WANs), validation of protocols,[11] etc. Every angle of this topic has been studied in great detail.

At the same time there has been considerable movement to standardize network architectures.[8] An environment in which all systems communicate with the same protocols is ideal. It allows maximum connectivity among systems and maximum interconnectivity among systems of different vendors, and removes impediments to network growth that network operators would experience if they had to spend time resolving disparities in protocols.

As a minimum, if one cannot guarantee commonality of protocols across all systems, it is useful to have a reference model to which all systems can be converted. One aspect of the Open Systems Interconnection (OSI) reference model is that it allows maximum interoperability among different network architectures. This is achieved not by having a common architecture, but by ensuring that gateways can be built to the standard reference architecture.

While much has been going on in the architecture and theory of protocols, few papers have provided insight into the structure of the systems that must implement these protocols. This oversight is unfortunate, for there is a close relationship between the particular protocols that are desirable in a specific environment and the particular machine for which the protocols are targeted. We attempt to substantiate this claim in this paper.

This relationship helps to explain why there are so many different architectures in use. Each vendor, while devising its protocols, has a different mental model of what the implementing system will look like. Thus, each architect is led to design very different protocols. Although these are not the only factors that govern the selection of protocols, they are important ones.

A result of these observations is an understanding of why it is difficult to expect a single standard to be "best" for all environments. We do not disagree with efforts to try; certainly there are many disparities that arise, for historical and arbitrary reasons. But many disparities can be explained by understanding the different mental models used by different architects. Once the reasons for disparities are understood, they can suggest ways to resolve those differences.

Another viewpoint is that communication systems should be designed to allow usage of the "best" protocols. This is a reasonable perspective to take. Unfortunately, it is rarely clear which protocol is best. Moreover, communication function must often be provided to existing machines. Finally, there are always design trade-offs among protocol design, system design, and cost. At best, we can expect protocols and communication systems to be built together, with the constraints of one influencing the design of the other.

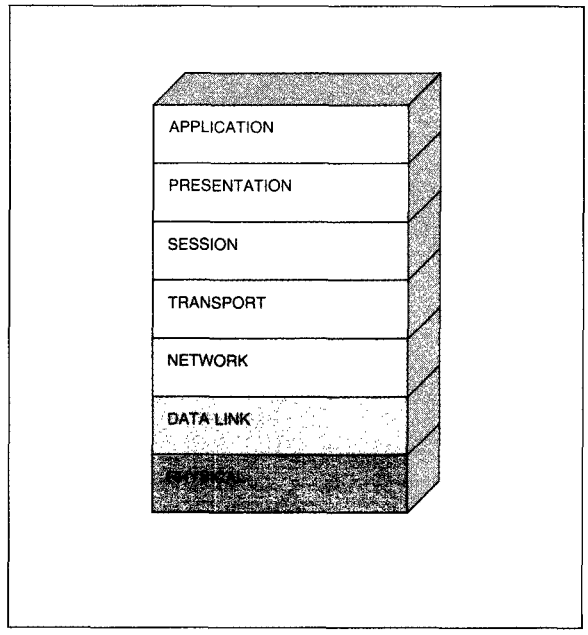
The purpose of this paper is to illustrate the close coupling between protocol design and communication system design. In particular, we discuss two areas in which the particular choices made for protocols should depend on the system, and vice versa. These areas are network control protocols and flow control protocols. Many more examples are available, but these serve as useful illustrations of the theme. With detailed examples, we substantiate the aforementioned close relationship.

Before developing this theme in detail, we first define the key terms used. In the following two sections we describe the interrelationships between systems and protocols, which is the principal point of the paper. In succeeding sections, we attempt to draw conclusions from this for standards activities and protocol design. Next we illustrate how the problem will become increasingly complex as the diversity of high-level protocols increases. Finally, the last section before the summary describes other issues that are important in protocol design.

Definitions. The two key subjects we discuss herein are protocols and communication systems:

- *Protocols* consist of all the rules that govern communications between computing systems, including rules at any of the seven layers of architectures such as Systems Network Architecture (SNA)[6] or OSI.[8] See Figure 1.
- A *communication system* is any system that implements protocols, including systems running end-user function such as mainframes, minicomputers, or workstations, or dedicated communication controllers such as IBM 3725s, or X.25 packet switch nodes (data circuit-terminating equipment, or DCE).

Figure 1 Seven OSI layers

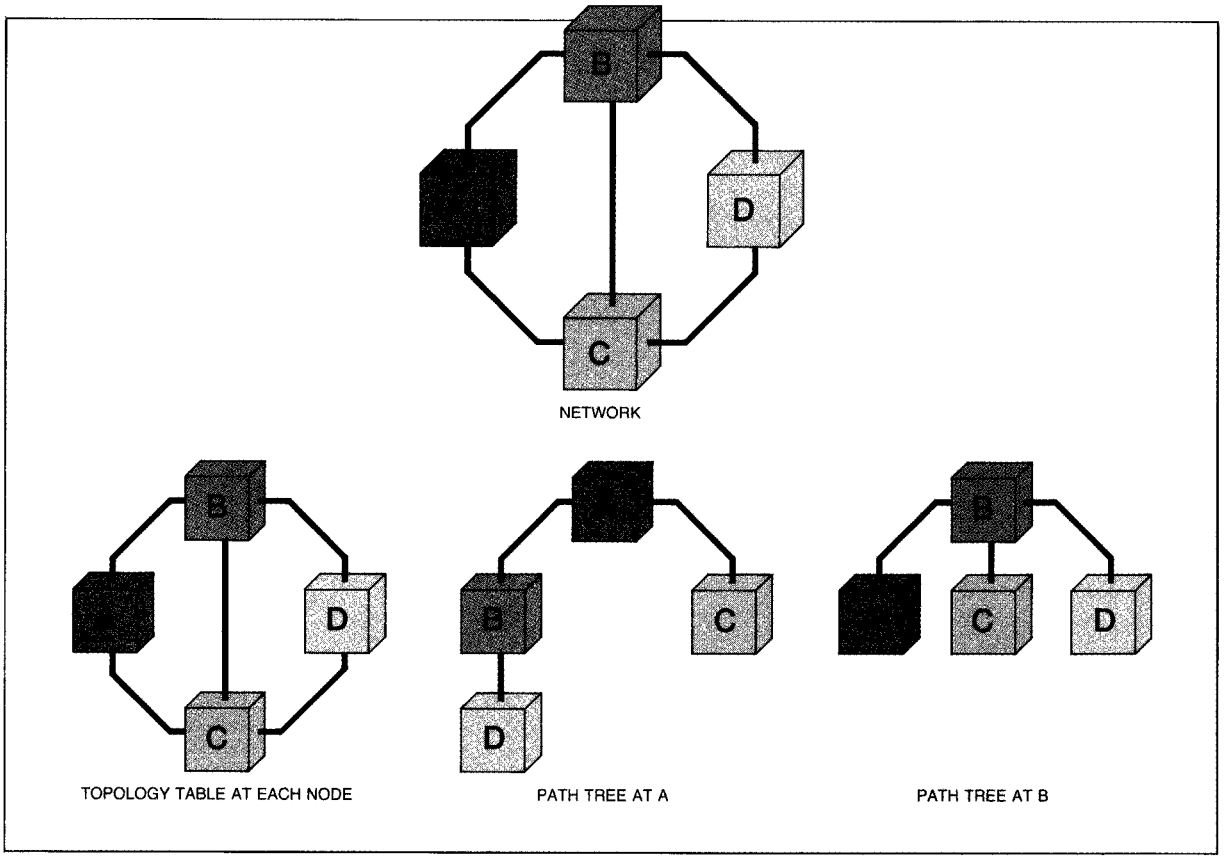


Network control: Protocols and controllers

In this section we describe network control examples that illustrate the relationship between communication systems and protocols. First we define network control by explaining that network protocols are conveniently divided into three classes:

1. *Data transmission protocols* involve the actual transmission of useful user data from one node to another. Protocols involved in this phase of network operation include steady-state intermediate node routing, flow control, session end-to-end acknowledgment, etc.
2. *Network control protocols* involve negotiation among nodes prior to useful user data transmission to allow data to be ultimately transmitted. Examples of these protocols include physical link activation, accumulation of network status to be used in route determination, and directories.

Figure 2 Topology approach



3. *Network management protocols* involve the observation of a running network, including session tracking, performance monitoring, and error logging.

The lack or possession of certain key features by a communication system may determine the selection of network control protocols. These features include the amount of main memory, the availability of secondary storage, the internal performance of the machine, the internal architecture of the machine, the machine instruction set, and whether the communication system runs user applications. These factors, taken in tandem with the perceived environment, are critical in determining "correct" protocols.

Routing example. An example is the routing control exercised to gather network status information needed to select optimal routes. Three common techniques are

- Predefined routes
- Distributed shortest-path algorithms
- Topology algorithms

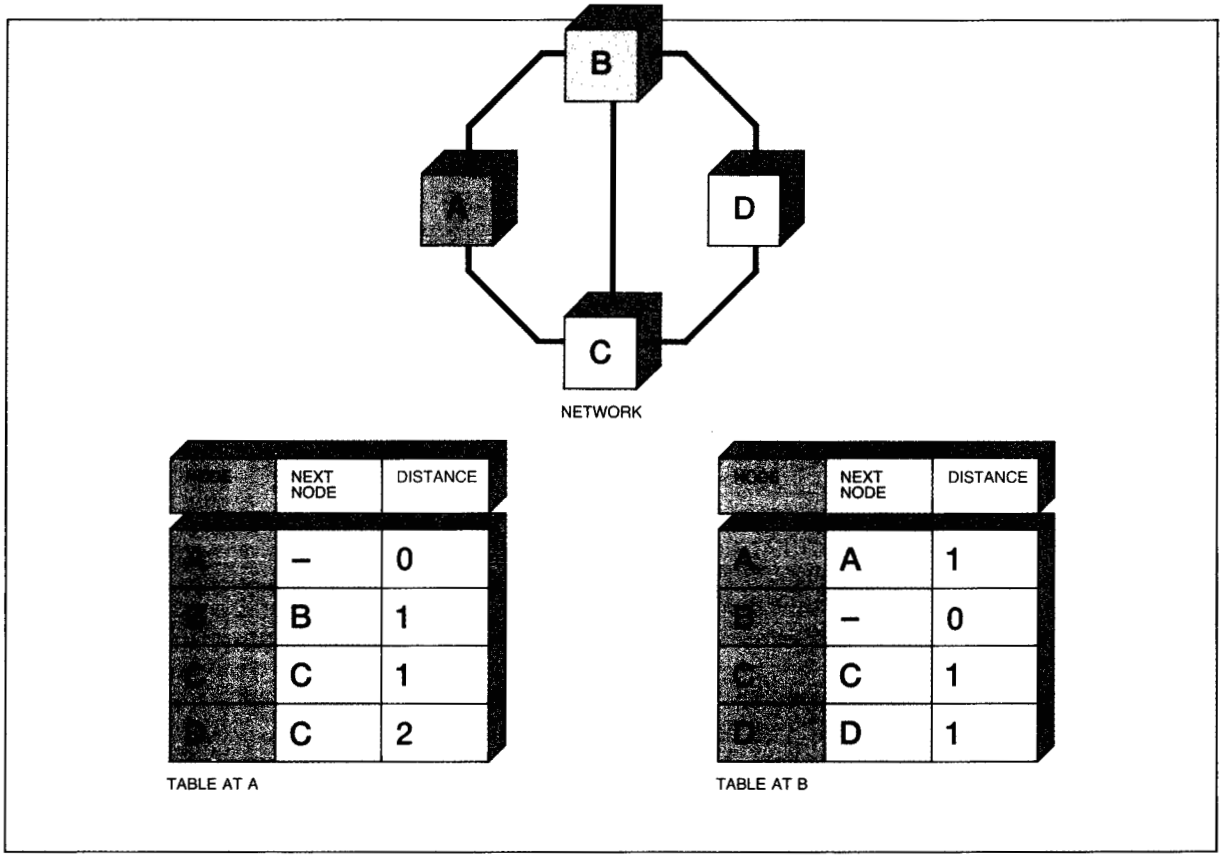
In *predefined routes*,[12] routes are established when the network is configured, and users are assigned to routes by pre-established conventions. These procedures are appropriate when communication systems lack real or virtual storage required by the other techniques or are limited in processing power. They are also more appropriate where the machine architecture makes it difficult to physically add new connections on dial lines (for example, the machine architecture might limit the number of devices that could be connected, or the bus speed of the machine could cause memory/processor overruns by adding additional lines/capacity).

Environmental factors could also lead to a decision to opt for predefinition. If network traffic patterns

are highly structured and time-invariant, the best performance is achieved by simultaneous planning of the topology and routes to be used. Also, in a single-LAN environment, the route from one node to another is determined. Hence, the desirability of expending network overhead to dynamically adjust routes is minimized.

Distributed shortest-path algorithms[13] and *topology algorithms*[14] are quite similar. Both collect network status information on a dynamic basis and use it to determine best paths. The methods are slightly different—the topology approach collects the entire topology at each node (Figure 2) and from this does a single-source shortest-path calculation, whereas the distributed shortest-path approach ensures that each node accumulates the *shortest paths* of its neighbors (Figure 3) to each destination so that it may determine its own shortest paths. Generally, one can use either as a desirable alternative to predefinition, yet the particular communication system may cause selection of one or the other. These techniques are generally desirable if the machines are "hot-pluggable," allowing frequent changes in topology. Also, if adequate storage exists, these techniques allow ease-of-use improvements, better availability and performance (such as factoring traffic congestion dynamically into the route calculation), and easier reconfiguration than predefined techniques. Since a topology data base can be reasonably complicated, these approaches place additional requirements on the machine architecture and capacity. Clearly, a machine architecture such as the Zilog Z80 processor, with a maximum data segment of 64K bytes of memory, will place a significant constraint on the effective processing of the topology data base. In addition, a larger topology requires greater processing power in the controller to perform the needed

Figure 3 Shortest-path approach



calculations effectively. Thus, the processing power and memory trade-offs chosen will affect the approaches taken.

To elaborate, the topology approach requires more storage than the distributed shortest-path approach. The reasons are that the actual topology data base can be large and the maintenance of the paths can be calculated from the topology. Thus, a storage-constrained system might prefer the distributed shortest-path approach. Even if adequate storage exists for the larger topology data base, it may be preferable to use the extra storage for buffering transit messages, rather than for the added topology information. Thus, a curious trade-off exists between a potentially better routing procedure and storage available for flow control. For situations with unlimited storage, the topology approach provides better control over potential paths and has better reliability characteristics.[14]

Even the machine instruction set can play a role in selecting a protocol.

Similarly, even the machine instruction set can play a role in selecting a protocol. In a large network in which the number of machine instructions calculating paths grows to be a significant processing load, one might choose the topology approach. This choice would permit the use of very efficient data structures to reduce the amount of time spent calculating paths. But this optimization is usable only if the machine instruction set allows convenient manipulation of such data structures. If a communication system has a primitive instruction set optimized to the store-and-forward transmission of messages, it will be unable to utilize such an optimization, and the topology approach loses an important advantage.

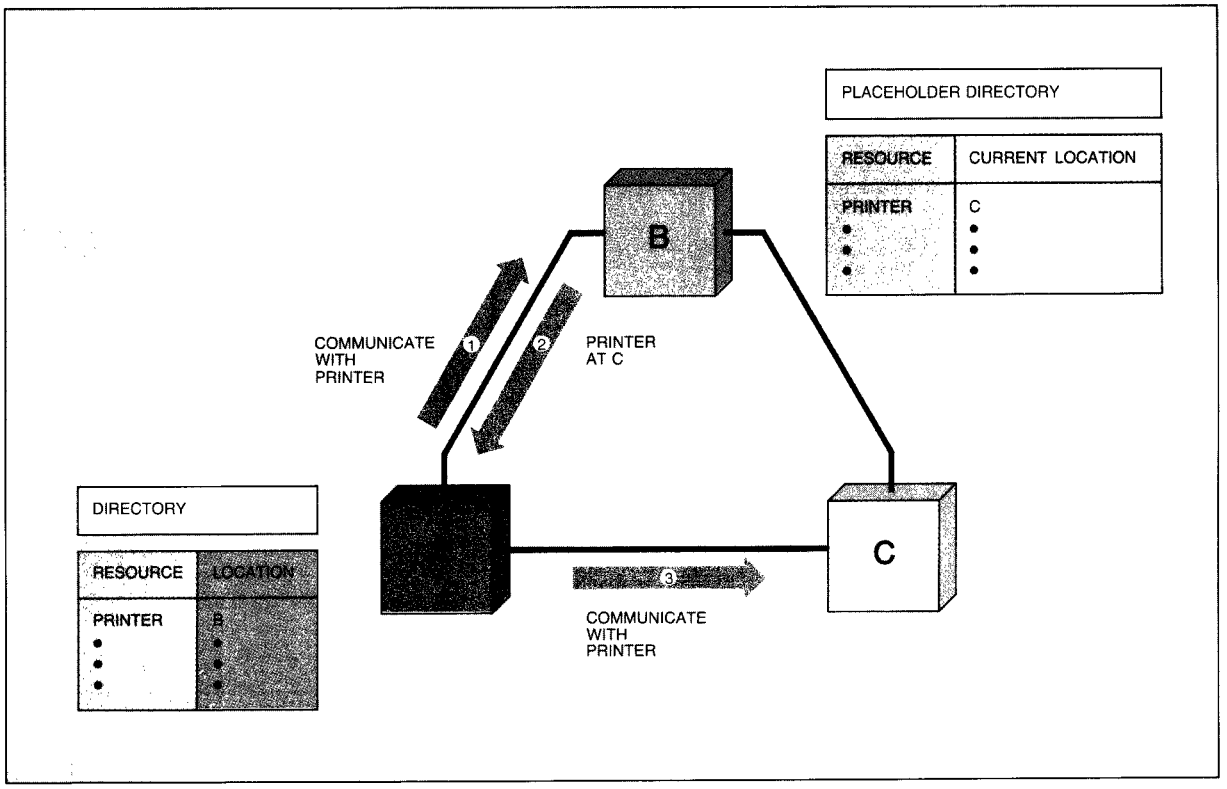
Function placement examples. A second network control example of how communication systems affect protocol design is the *placement* of network control. Network control decisions (such as link activation, network directory, route selection) can be fully distributed, centralized, hierarchical, or hybrids of various forms of simpler architectures.

Network control decisions are made on the basis of information in various data bases. Thus, the amount of storage as well as the quality of data base management determines which system can play which roles in controlling the network. The topological position of these systems and the question of whether *all* systems can participate in network control decisions determine whether to centralize or distribute control. For example, SNA[6] places topological restrictions on less capable PU-T2 nodes.

Another determining issue for network protocol placement is whether the end-user sessions are actually running in the same physical machine as the communication system. For reasons of authorization and organizational control, it may be important that directory and descriptive information (such as ownership, security requirements, degree of concurrency) about the network resources exists in the same node in which the resources are located. In a network where each communication system is a node with resources, it is reasonable for each node to implement a distributed network search protocol[15] to locate resources in a peer manner. In contrast, if only a portion of the nodes support users and applications and the others only provide intermediate node-switching function, it would be inconvenient for the pure switching nodes to participate in such search algorithms. This situation leads to different protocol implications for such a networking architecture.

Update frequency example. A related issue is the degree of currency of network directories. This issue is determined by the types of communication applications running in the systems and the storage/cycles available. Suppose a service or endpoint application moves from one node in the network to another (e.g., dynamic placement based upon usage). One could broadcast the change if movement were infrequent; alternatively, one could employ techniques from the distributed data world, namely, put a *placeholder* in the node where the application originally resided that points to where it moved (see Figure 4). In this case, when another node requested the application, it would go to the original node *and then* discover the new location and consequently update its directory data base in the process. Thus, the updating of the directory data base either can happen in real time if movement of applications/services is

Figure 4 Placeholder approach to network directories



infrequent or can be deferred until service is actually requested. This choice is related both to the frequency of application movement and the ability to keep up (from a cycles viewpoint) with potentially frequent directory updates.

In summary, network control algorithms organize a great deal of information about the network. However, the information accumulated, the location of this information, and its interpretation are handled differently on the basis of machine architecture. This in turn must have implications for the protocols implemented.

Flow control protocols

An excellent example of the relationship between communication system design and network architecture and protocols is the area of flow control protocols. Flow control protocols regulate the flow of traffic in networks to meet the following objectives:

1. Deadlock freedom
2. Good response time/throughput trade-off
3. Minimal unfairness
4. Efficient buffer utilization
5. Low overhead

Occasionally these goals are in conflict!

Deadlock avoidance example. The first example deals with whether flow control should have deadlock avoidance as its primary objective. Certain schemes prevent store-and-forward deadlock,[16] at the cost of increased overhead, whereas others utilize controls that statistically reduce the probability of deadlock without prevention.[17]

When a communication system has limited storage, the probability of deadlock is high (see Figure 5). Deadlock occurs in systems that are also supporting user applications; the system does not allocate sufficient storage to communications. A deadlock-free protocol is a proper design decision in such environments.

As the amount of storage available to communications increases, the statistical probability of deadlock

Figure 5 Deadlock with limited buffers (six in example)

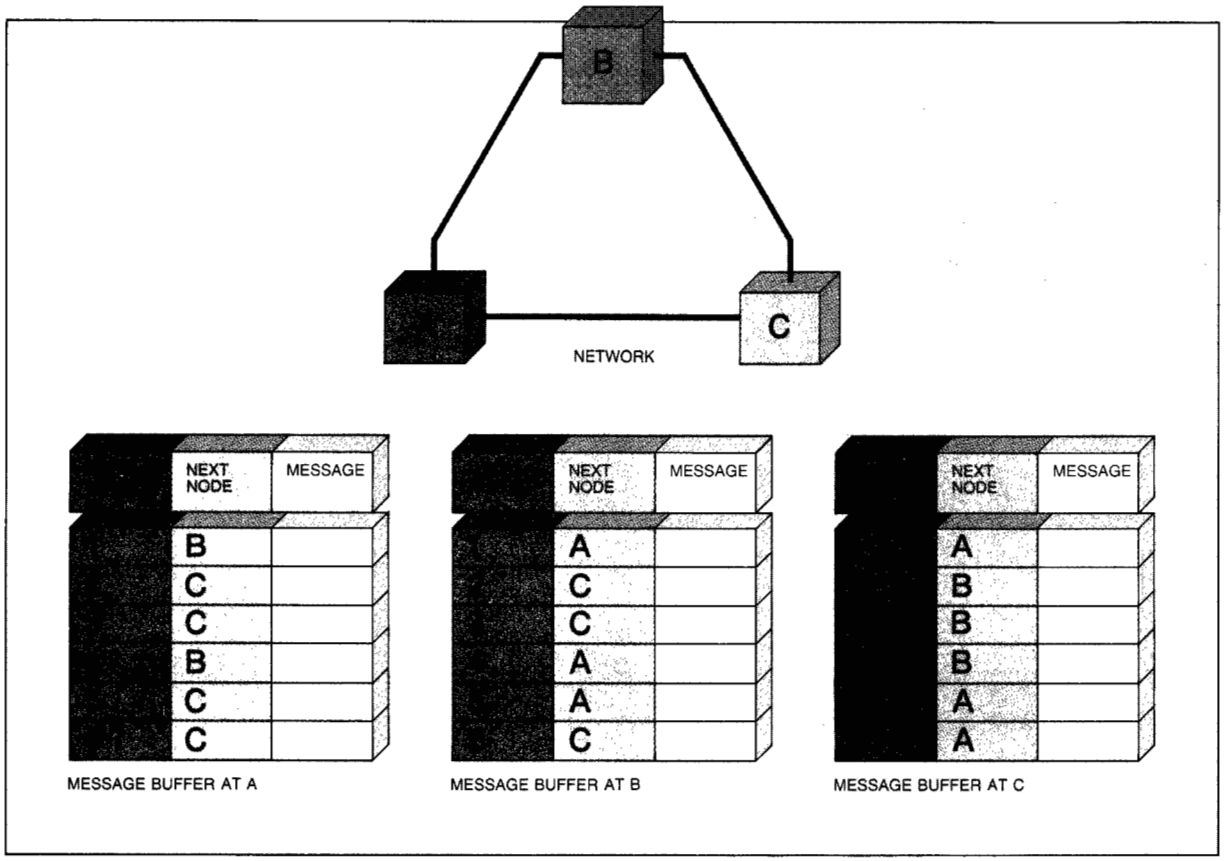
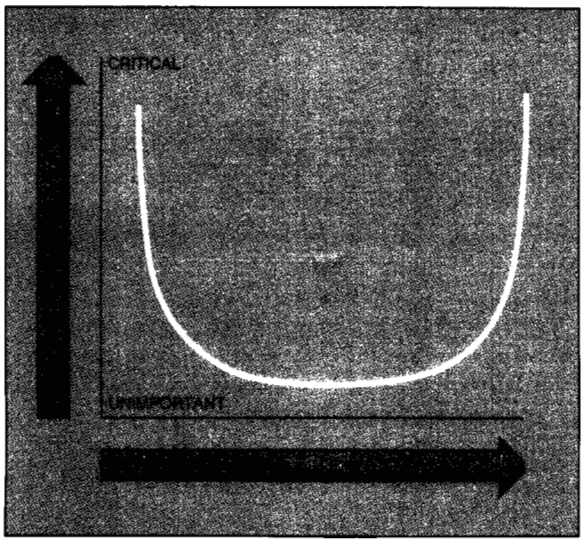


Figure 6 Storage versus deadlock prevention



diminishes. In that case, a deadlock-prevention mechanism may be undesirable. For example, the session-by-session buffer reservation technique[18] restricts sessions from entry into the network. Alternatively, the buffer-class deadlock-prevention technique[16] causes unfairness by requiring certain messages to be blocked due to congestion at other nodes. An attractive alternative is a less restrictive scheme such as SNA virtual route flow control,[17] which has probabilistic deadlock avoidance.

As storage increases further, a deadlock-free scheme such as reserving buffers on a per-session basis is appropriate. Here, one is no longer concerned with restricting the number of sessions entered into the network; an adequate quantity can enter. Thus, a deadlock-free scheme is not restrictive, but it serves important purposes. It prevents a session from grabbing so much network capacity that the network becomes congested and deadlocked due to an "out-of-control" session.

Figure 6 illustrates the paradoxical relationship between the amount of storage and the importance of deadlock-free protocols. They begin as being very important, but this importance decreases until they reach relative insignificance, at which point they once again grow in importance.

Satellite effect on flow control. Another flow control example relates to satellite support. The existence or absence of satellite support, which is a machine architecture issue, affects flow control procedures in many ways. A key item is whether flow on a route is managed on a route basis or a "hop" basis.[19]

Generally, when flow is regulated on a route basis using "window" algorithms, the source node has an allocation of permits or "window" of messages that can be sent before receiving an acknowledgment. The window size should be large enough to allow good throughput (no excessive waits for acknowledgments) but low enough to prevent congestion in intermediate nodes. When flow is managed on a hop basis, the same protocol applies, but separate windows and acknowledgments operate over each link in the route.

A good protocol for networks in which controllers lack satellite support is to choose route-level management because of lower overhead (fewer acknowledgments—see Figure 7). Unfortunately, if satellite links exist in the network, large windows are required to "fill the pipe" because of satellite propagation delay (Figure 8). In this case, use of route-level windows causes considerable network congestion. However, if hop-level windows are used, the large windows can be isolated to the satellite link, thus avoiding excessive network congestion (Figure 9).

Figure 7 Route level; window equals 8

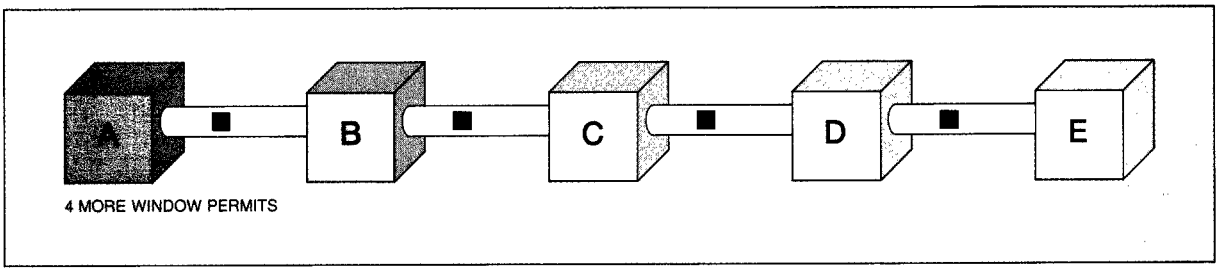


Figure 8 Route level with satellite; window equals 25

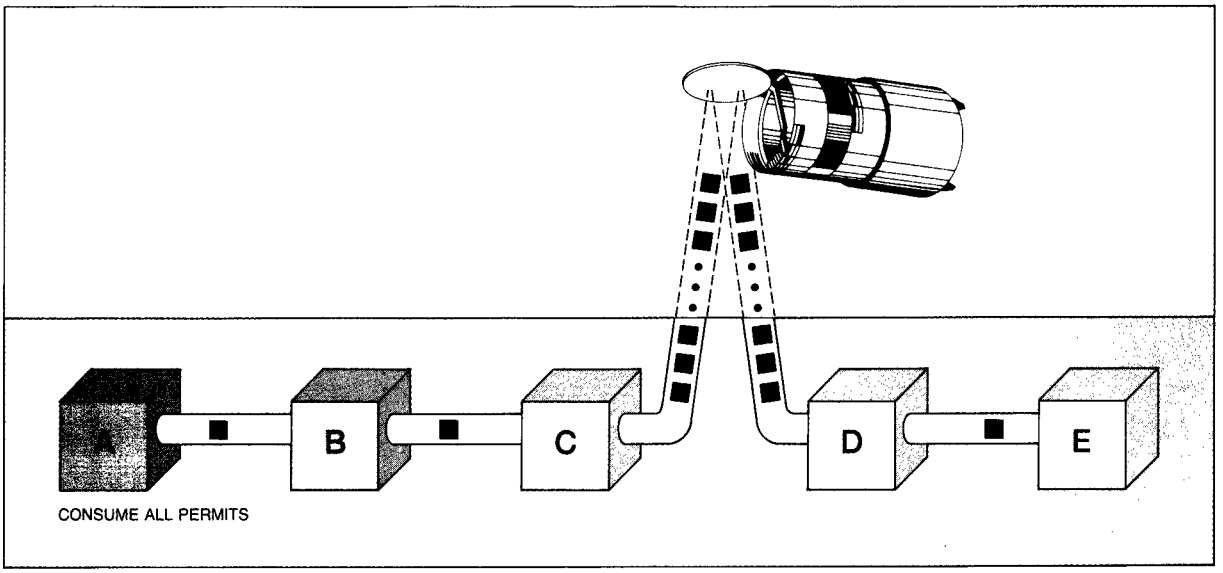
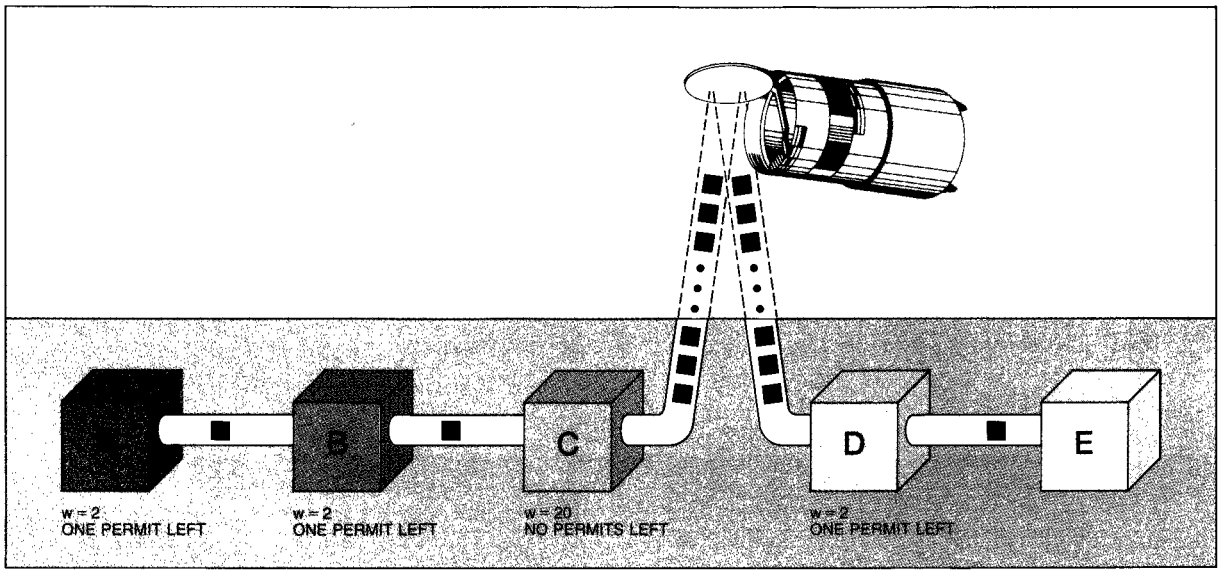


Figure 9 Hop level; window equals 20



Satellite transmission example. A lower-level example is in the *link retransmission protocol.* Two typical protocols are *go-back-N* and *selective reject.* In go-back-*N* protocols, when the receiver detects a transmission error, it instructs the transmitter to retransmit the packet in error as well as any subsequent packets. In selective reject, the receiver requests retransmission only of the packet in error. It turns out that communication system design (in particular, the maximum-speed satellite link it supports) affects the relative desirability of using these protocols.

The key advantage of go-back-*N* is that it requires buffering only at the transmitter—the receiver immediately transmits successful packets. The protocols are also simpler to implement and are preferred in most environments. Selective reject has superior link utilization properties at large window sizes and high error rates.

For a communication system capable of supporting low-speed satellite connectivity, go-back-*N* schemes are most appropriate, since relatively low window sizes are required to obtain high utilization. However, the selective-reject protocols are appropriate for systems that allow higher-speed link support.

The amount of storage is also an issue. Selective reject requires storage at the transmitter (in case retransmission is necessary) and at the receiver (for cases in which an error occurs and correct packets must wait for a retransmission). Selective reject cannot be used if insufficient storage exists.

The internal communication system structure is also an issue. If the storage for satellite buffering is shared among all links connected to a node, there may be (statistically) adequate storage for selective reject. However, if the link protocol is on specialized satellite adapter cards (to off-load this cycle-consuming function), the availability of storage is more constrained.

Designing a communication system for maximum protocol flexibility

In previous sections we have argued that the best protocol for a particular function depends on the machine architecture of the communication system implementing the function. In this section we discuss whether the examples cited previously are relics of misguided system designs or reveal inherent difficulties. It might be argued that by observing technology trends one may define the "best" protocols for all future systems. Our contention is that the need to service different environments, the need to examine competing objectives, requirements to reduce cost, and uncertainty about technology trends will continue to favor different protocols for the same func-

tion in different future implementations. As before, this viewpoint is supported by examples.

The first example illustrates differences in the ideal design in directory structure due to differences in types of computing and communication systems. One environment that exists today and will exist in the foreseeable future is that of the mainframe. In that environment, large hosts, which control most of the network resources and have cycles, storage, and data base support, are logical repositories for network directories. Other communication systems such as dedicated communication controllers[20] provide a front-end for the host, off-loading specialized functions of line termination and message formatting. Thus, directories will be centralized or managed hierarchically by mainframes on the periphery of the network.

The personal computer environment is another one that exists today and will exist in the future. Here, large numbers of workstations need a communication server (e.g., a minicomputer) to provide network management. A logical directory protocol is a distributed protocol between peer (mesh-connected) servers. Alternatively, if the network were also structured on departmental lines, it would suggest a hybrid, an example of a set of protocols that have to factor in both distributed and hierarchical searches, depending on the location of the systems in the network. A given communication system might have to support both sets of protocols depending on the direction of the request.

The next example illustrates how uncertainty about technology trends can perpetuate protocol disparities for flow control/link control. Many protocols require buffering per link/route as a function of line speed. Assuming an infinite amount of storage, one can devise the ideal protocol. Moreover, since storage has gotten relatively inexpensive, one may imagine that such an ideal protocol should always be used for the future. Unfortunately, high-bandwidth communication is also becoming inexpensive. This development, coupled with the fact that *any* system design must be used for some number of years, may cause one to re-evaluate the assumption of infinite memory. It may be too inconvenient to regularly increase the memory on line adaptors each time higher-speed lines are needed. Thus, the proper engineering solution may differ from the ideal solution.

Next we point out that communications must be a part of a wide variety of different types of communicating systems. Clearly, the design points of mainframes, minicomputers, workstations, terminals, supercomputers, terminal controllers, private branch exchanges (PBXs), dedicated packet switches, etc. are different. This will force different system designs and different degrees of desirability for different protocols.

Finally, the requirements to reduce costs and make cost trade-offs will always encourage designers to optimize machines differently. In any design there is some cost component that dominates. The component may be memory, data base support, machine cycles, ability to distribute function to other processors, software development cost, etc. Whichever factor dominates will suggest a design optimized by reducing the need for that factor and will suggest protocols designed around that optimization. Different systems, optimized differently, will require different protocols.

Different systems, optimized differently, will require different protocols.

Implications for standards

There are three *wrong* conclusions that one could derive from the above discussion:

1. There is no value in standards—they cannot hide inherent differences.
2. There is no hope for standardization—there cannot be agreement among those with different design points.
3. Successful standards are bad—they force nonoptimal solutions.

All of these potential conclusions are invalid.

One must understand the purpose of standards before rushing to conclusions about them. There are three important reasons for standards:

1. They achieve uniformity where there is no excuse for diversity.

2. They encourage interoperability.
3. They provide a reference model.

To elaborate, Point 1 above recognizes that in certain situations diversity will prevail. It is important to acknowledge this and to allow that diversity. For, by pigeonholing different factors together, one will experience bad performance, difficulties in achieving agreement, and occasionally an inability to accommodate the standard. However, there are many more instances in which diversity is not inherent, or in which differences are the fields in a header, the value of a parameter, a detail of a protocol, or even totally different protocols to solve essentially the same problem. These cases must be recognized. And there is great value, hope, and good performance in standardizing here. *For if standards are not achieved, the greatest cost in future systems will be in software development to convert from one to another among protocols.*

Point 2 recognizes that there will be different *network architectures*. Standardization cannot have as its goal the creation of a single universal network architecture. It would lead to the conclusions mentioned above. However, there is a requirement for all systems to communicate with one another. Such communication can be achieved by having a common interface to which all sets of protocols can map.

Point 3 merely indicates that different designers should use the same vocabulary. Common terminology is a prerequisite to achieving further agreement.

We believe that all of these points are implicitly understood by most designers of protocols and standards. We make these points explicit:

1. To focus designers' attention on these issues
2. To encourage the most realistic view of standards
3. To clarify these issues to the uninitiated by providing perspective

Our proof that designers are aware of these issues is that they are already taking actions which are consistent with this view, as discussed in the next section.

What to do about diversity

Base and towers. The ideal situation is to recognize the total set of functions needed in the most complex case and a minimum set of functions needed for simpler cases. The minimum functions, called *base functions*, are required of all systems. Various *towers* representing additional functions are defined. Each node implements whichever high-level towers it desires, as well as lower-level towers (and the base) needed to support the higher-level towers (see Figure 10). When nodes communicate, they negotiate their capabilities to achieve the greatest common subset of their towers. As a minimum, function is always available at the base level.

Often, commonality is appropriate at the internetwork level.

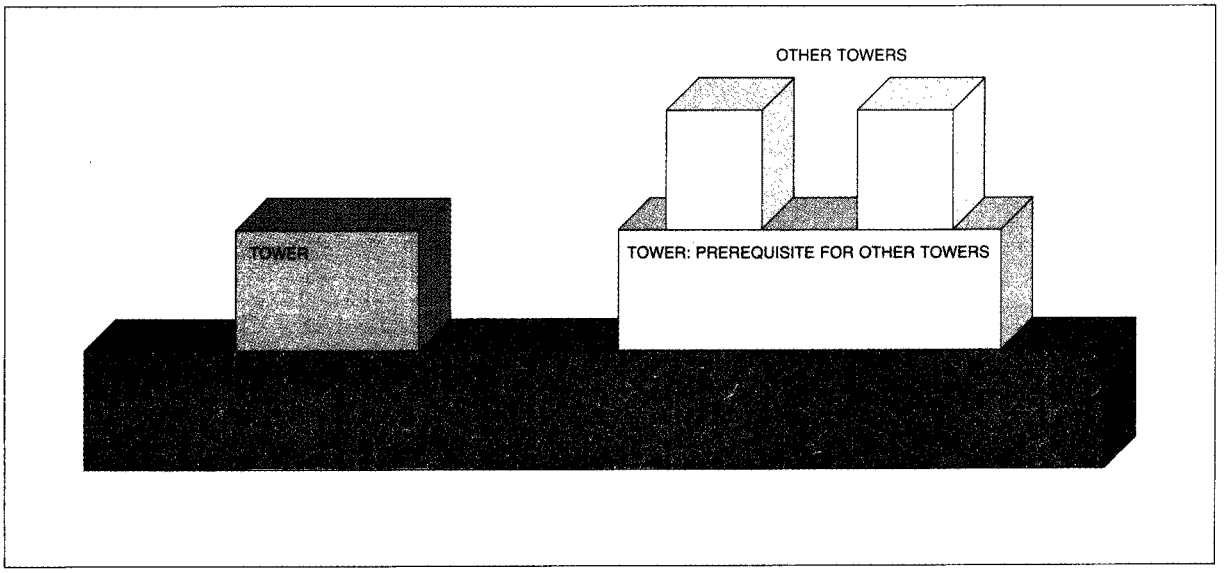
This feature is a key aspect of APPC (an SNA session protocol[21]) and the Advise directory system,[22] as well as certain functions on OSI level 4.[23] It would be more widespread if it were not often difficult to achieve.

Different modes. When there is no commonality among different protocols for similar function, it is important to categorize the key differences. Categorization allows one to gain a clear understanding of where differences should exist and allows ease of standardization within each category. The best example of this is the work of the IEEE 802 Committee to standardize different local-area network protocols.

Different networks. Often, commonality is appropriate at the internetwork level. This commonality is trivially illustrated by the fact that different networks have different architectures. But what is perhaps more interesting is that even within one vendor (IBM) there are different routing architectures (both subsets of SNA) for different systems. The *SNA backbone* provides routes in a mesh configuration, whereas the *SNA terminal subnetwork* allows hierarchical connectivity to a backbone node. At the border between networks, the *SNA boundary function* provides the required conversion.

Similarly, much of the OSI literature explicitly recognizes the key role of OSI in conversions between networks supplied by different vendors.

Figure 10 Base and towers



Impact of high-level protocols

This paper has focused on low-level protocols. However, communication systems also participate in high-level protocols to provide transaction switching, transparent accesses to services, and facilities for interprocess communication between coupled processors. This section outlines higher-level protocol issues.

Consider the fundamental requirement for *transaction switching*. Transaction switching places a new requirement on the communication system to determine where to route a message on the basis of the semantics of the message, i.e., to determine the proper target for the message by using information provided in the message. This requirement implies that there is a need for a mapping services layer that will scan the message for a transaction code and, by means of the code, determine the proper target location. An effective transaction management system also implies requirements on the operating system, data base management, etc.

In the world of voice, there is a similar requirement, namely, to determine the proper target communication systems on the basis of a dialed-in telephone number. The problem is further compounded if one considers the problem of dialing "800" numbers. The problem is "Where is the 800 number?" Given multiple telephone operating companies (and Local-Access Telephone Areas, or LATAs) and multiple common carriers, the question is really: "Is the 800 number *within* the LATA that I am dialing from? Is it in one of the many common carriers that the LATA is connected to? Or is it in a LATA of another operating company connected to a common carrier?" These are nontrivial questions.

High-level protocols are needed in both cases to determine the proper routing destination. The problems in the voice world are further compounded because of the high probability that the communication systems employed by one operating company will be different from those used by the common carriers and those used by the local exchanges.

Clearly, standardization is critical in the voice world in order to solve the above problem. In the data world, although it is true that interconnection between communication systems of different vendors is both possible and, in some cases, desirable, such systems are generally local to a given vendor. Furthermore, no two transaction subsystems are identical or interchangeable; each has its own form of transaction definition and mapping services. This is not to say that there should not be a common set of mapping services which could be employed by the communication system such that, given the arrival of a message, and a stored template, the communi-

cation system could not determine the appropriate destination. As with many applications developed over the years, transaction management subsystems have evolved in a somewhat *ad hoc* manner.

There are two alternative solutions to this question of transparency.

Next we consider the need of the applications to gain access to the services of the network. We encounter a different set of issues:

- Should the application programmer be aware of the location of the service (such as a file server or print server)? There are two alternative solutions to this question of transparency:[21,24,25] By transparency, the application programmer assumes that all services are local, with remote being handled transparently by the conjunction of the operating system and the communication subsystem; or, by transparency, we assume a new naming convention that fully qualifies each object and service by its name and location (this form of transparency optimizes for the service being remote). The former implies a significant requirement on the communication system to perform high-level distributed directory searches.

  Just as we have discussed the "800" problem in the voice world, we have a similar problem in dealing with object transparency. If an application asks for access to a file named SYSJOUR, the question, in a general network, of "Where is the file?" is a nontrivial one. Clearly, the *system* could first search to see if the file were local to the requesting node/LAN; if not, it could resolve to a very complex distributed search (assuming a nonhierarchical view). In today's systems, a clear separation exists between communication management and object management (such as a file system). In addressing transparent access to objects, the directory management function of the communication system would have to be augmented to include information about the location of objects.

  Moreover, consider that the names of objects in the network may not be unique. This is analogous to the synonym problem associated with 800 numbers in the voice world. Suppose one had an 800 number by the name of 800-MOMADUK. What would happen if two different telephone operating companies allowed their customers to freely assign synonyms (just as two different states might assign the license plate HAPPY to two different drivers)? If 800-MOMADUK were assigned to a trucking company in one LATA and assigned by a different operating company to a restaurant in another LATA, which of the two businesses would a casual caller be connected to if the call originated in a LATA owned by a third telephone company? Solutions not only require greater emphasis on standards but also impact the directory services of the communication system (if allowed). The analogy here refers to objects having the same name but existing in different nodes. For example, suppose there are two or more different file servers in the network; what is the probability that there will exist files with the same names but located on different nodes (and containing very different data)? The question is "Which file will a casual application get when attempting access and being located at an arbitrary node in the network?" Furthermore, resolution of the issue of "right of access" becomes extremely complex.

- Given a heterogeneous environment, there exists the issue of mapping between different file systems and object managers in the network (e.g., between ASCII and EBCDIC encoding schemes; linear byte space versus logical record formats; field definitions between different data base management systems; etc.).

- Performance and bandwidth of the media and communication systems: If one were to use the network for transparent access to service, it would never be used if the performance were significantly bad. A rule of thumb is that the performance of access to services/data in the network should be reasonably close to access to services/data in the requesting node; e.g., access to a file on a remote file server should take approximately the same time it takes to access the file on a local disk. If performance is significantly slower, then the user will not use the transparent access and will, alternatively, use application-to-application communication protocols, making conscious decisions on the appropriate function split between nodes.

Finally we consider the question of *interprocess communication* (IPC). Here the real question is "Why does an application programmer want access to the network?" While it is true that the application programmer does, in some instances, want the capability of interprocess communication (IPC), it is not a general requirement. Typical applications want the ability to access the services offered by the network without the added complexity of IPC. Most high-level languages, in fact, do not offer primitives for communication. A typical application programmer learns the basics of the language and struggles through the mechanisms for reading and writing data to files and talking to the terminal.

Interprocess communication is nontrivial. The application programmer has to worry about

- Concurrency (who starts first, who initiates the conversation, how do two applications synchronize, etc.).
- How do two applications synchronize termination? This is a very hard problem. Solutions such as use of the Byzantine Agreement are nontrivial and impact the operating system, transaction management subsystem, and communication subsystems. The solutions can be classified but are certainly not standardized and are dependent upon the characteristics of the conversation (recoverable versus nonrecoverable, the locking protocols employed, whether or not there is shared memory, etc.).

Most application programmers code sequentially; to have to worry about asynchronous events and concurrent programs is not only difficult but extremely challenging to prove correctness.

Additional factors in controller/protocol design

Although protocols depend on the communication systems that implement them, both are also affected by other factors:

1. *Link transmission speeds.* Many items, including flow control strategy, depend on link speed because the amount of outstanding information "filling the pipe" depends on the size of the pipe. Also, the amount of complexity one places in a protocol depends on line speed. If speeds are slow, one can afford to provide node-by-node processing for segmentation, retransmission, and error recovery, and not affect the order of magnitude of performance. At higher data rates, protocol complexity can cause nodal processing to interfere significantly with high-speed data transport. In addition, very high bandwidth will place additional requirements on the communication controller bus and its connectivity to the media.
2. *Probability of transmission error* (e.g., transmission over telephone wires at 300 baud has significantly greater probability of error than transmission over optical fibers). If one assumes a very high probability of error, the protocols have to be concerned with handling such situations on a frequent basis. If one assumes that errors are relatively improbable, communication protocols will be optimized for lack of error (with an error handled as an exception case and performance at times of errors irrelevant).
3. *Openness.* If the protocol provides an application program interface to be used by general applications written by application programmers, the communications software must guard against either accidental or malicious misuse of the interfaces. Thus, the complexity of the communication interfaces can vary significantly based upon whether the interfaces are exposed to general applications or remain as purely internal interfaces. In addition, the machine architecture has to guarantee the integrity of the operating system and the communication subsystem against the accidental/overt action of the general applications.
4. *Communication locality of reference.* Are communicating entities likely to have topological proximity? In a very large corporate network involving LANs, gateways, and multiple mainframe "glass houses" over wide geographic distances, there will most certainly be locality of reference in that the majority of traffic in a LAN should be between processors within the LAN, and in decreasing order to the glass houses/departmental servers and then outward. Alternatively, in a university with a LAN, with much of the activity oriented toward request of services from one or more universities with multiple computers, we will see a very different picture of locality. The choices for distributed directories and the algorithms for finding services/users can vary according to the locality or lack of locality.
5. *Leased or switched facilities.* Clearly, if leased lines are employed, the protocols for establishing optimal (or near-optimal) paths are controllable. In contrast, if the lines are not controlled by the communication systems, path allocation will not be optimal. This is not to say that the problem does not exist or goes away; rather, the problem

is not addressed by the communication system, but by the common carrier.

6. *Voice or data or both.* In the past, the problems of the voice world have always been viewed as different from those of the data communications world. A relative truth, here, is that they are not qualitatively different; they merely occupy different places on the same spectrum. There are different recovery aspects and performance aspects in the sending or receiving of voice or data.

Summary

The design of protocols to govern communications between computing systems continues to evolve. Different network architectures continue to perform analogous functions differently, and within architectures (such as SNA and OSI) there are different allowable modes. These differences cause difficulty in accommodating heterogeneity in operating networks.

The first step in dealing with this problem is to appreciate its root causes. This paper argues that a key, inherent root cause is differences in systems that implement these protocols. Several examples are given to illustrate why this is true today and will remain true in the future.

Our hope is that an appreciation of these facts will help to increase momentum toward standardization and to increase awareness of potential pitfalls. The best standards will be achieved when we understand the benefits and the limitations of standard architectures.

Acknowledgment

The authors acknowledge helpful comments from Paul Green.

Cited references

1. P. E. Green, Jr., Editor, *Computer Network Architectures and Protocols*, Plenum Press, New York (1982).
2. *Proceedings of IEEE Infocom* (1984, 1985, 1986).
3. *Proceedings of IEEE Globecom* (1984, 1985).
4. *Proceedings of Data Communications Symposiums* (1985).
5. *Proceedings of ICC* (1985, 1986).
6. *Systems Network Architecture—Concepts and Products*, SC30-3112, IBM Corporation; available through IBM branch offices.
7. Green, *op. cit.*, Chapter 10.
8. *Open Systems Interconnection (OSI), Basic Reference Model*, Draft Proposal IS/DIS 7498, International Organization for Standardization, Geneva, Switzerland.
9. J. M. Jaffe and F. H. Moss, "A responsive distributed routing algorithm for computer networks," *IEEE Transactions on Communications* **COM-30,** No. 7, 1758–1762 (July 1982).
10. W. Bux, "Local-area subnetworks: A performance comparison," *IEEE Transactions on Communications* **COM-29,** No. 10, 1465–1473 (October 1981).
11. Green, *op. cit.*, Chapters 20–25.
12. Green, *op. cit.*, Chapter 11.
13. J. M. McQuillan, G. Falk, and I. Richer, "A review of the development and performance of the ARPANET routing algorithm," *IEEE Transactions on Communications* **COM-26,** 1802–1811 (December 1978).
14. J. M. McQuillan, I. Richer, and E. C. Rosen, "The new routing algorithm for the ARPANET," *IEEE Transactions on Communications* **COM-28,** No. 5, 711–719 (1980).
15. A. E. Baratz, J. P. Gray, P. E. Green, Jr., J. M. Jaffe, and D. P. Pozefsky, "SNA networks of small systems," *IEEE Journal on Selected Areas in Communications* **SAC-3,** No. 3, 416–426 (May 1985).
16. A. Giessler, A. Jagemann, E. Maser, and J. O. Hanle, "Flow control based on buffer classes," *IEEE Transactions on Communications* **COM-29,** No. 4, 436–444 (1981).
17. F. D. George and G. E. Young, "SNA flow control: Architecture and implementation," *IBM Systems Journal* **21,** No. 2, 179–210 (1982).
18. L. Tymes, "Routing and flow control in TYMNET," *IEEE Transactions on Communications* **COM-29,** No. 4, 392–398 (1981).
19. B. Kadaba and M.-S. Chen, IBM Research Division, private communication.
20. *IBM 3725 Communication Controller—Principles of Operation*, GA33-0013-5, IBM Corporation; available through IBM branch offices.
21. J. P. Gray, P. J. Hansen, P. Homan, M. A. Lerner, and M. Pozefsky, "Advanced program-to-program communication in SNA," *IBM Systems Journal* **22,** No. 4, 298–318 (1983).
22. I. Gopal, A. E. Baratz, and P. Kermani, "The ADVISE directory architecture," in preparation.
23. *Open Systems Interconnection, Basic Reference Model*, Level 4, Draft Proposal IS/DIS 7498, International Organization for Standardization, Geneva, Switzerland.
24. F. N. Parr, J. S. Auerbach, and B. C. Goldstein, "Distributed processing involving personal computers and mainframe hosts," *IEEE Journal on Selected Areas in Communications* **SAC-3,** No. 3, 479–489 (May 1985).
25. B. C. Goldstein, A. R. Heller, F. H. Moss, and I. Wladawsky-Berger, "Directions in cooperative processing between workstations and hosts," *IBM Systems Journal* **23,** No. 3, 236–244 (1984).

Barry C. Goldstein *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Goldstein is the Director of the Communication Systems Department in the Computer Sciences Department at the Research Center. In this position his responsibilities include performing advanced technology research into communication strategies and the role of small systems in the realm of distributed cooperating heterogeneous systems. Dr. Goldstein joined IBM in 1969. He received his B.S. degree in engineering science from New York University and his M.S. and Ph.D. degrees in computer and information sciences from Syracuse University.

Jeffrey M. Jaffe *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Jaffe has been a Research Staff Member in the Computer Sciences Department at the Research Center since 1979. He received a B.S. in mathematics, an M.S. in computer science, and a Ph.D. in computer science from the Massachusetts Institute of Technology in 1976, 1977, and 1979, respectively. He is currently the department manager of the Communications Systems Department. In 1984 and 1985 he spent a sabbatical year at the IBM Scientific Center in Haifa, Israel. His work for the past several years has been in the area of network architecture and protocols, in particular in the area of distributed routing algorithms. In 1982, Dr. Jaffe received an Outstanding Innovation Award for his work in dynamic routing. He has also received two divisional awards and two patent awards within IBM. He is a member of ACM, IEEE, and Phi Beta Kappa.

Reprint Order No. G321-5290.