A perspective on Advanced Peer-to-Peer Networking

by P. E. Green
R. J. Chappuis
J. D. Fisher
P. S. Frosch
C. E. Wood

This paper is intended to familiarize the reader with the many reasons for undertaking the design and implementation of peer networking on small and intermediate business machines such as the IBM System/36 family. Such networking function was recently announced as IBM's Advanced Peer-to-Peer Networking (APPN) on Release 5 of the System/36. This paper sets the stage for a companion paper in this same issue, which discusses the implementation experience and details of the System/36 APPN product. In the present paper, the history of System/36 communication is first reviewed, and it is shown how APPN was a natural evolution from earlier function. Then an extensive study of user requirements that was started in 1982 is summarized. The paper concludes with a brief technical tutorial on the structure of the APPN design.

In June 1986, IBM announced a new concept in networking for small computers, called *Advanced Peer-to-Peer Networking* (APPN). It was implemented on the business machines of the System/36 family,[1] a series of intermediate processors that have featured such distributed processing functions as remote logon, document distribution (SNA Distribution Services–SNADS),[2] and Distributed Data Management (DDM).[3]

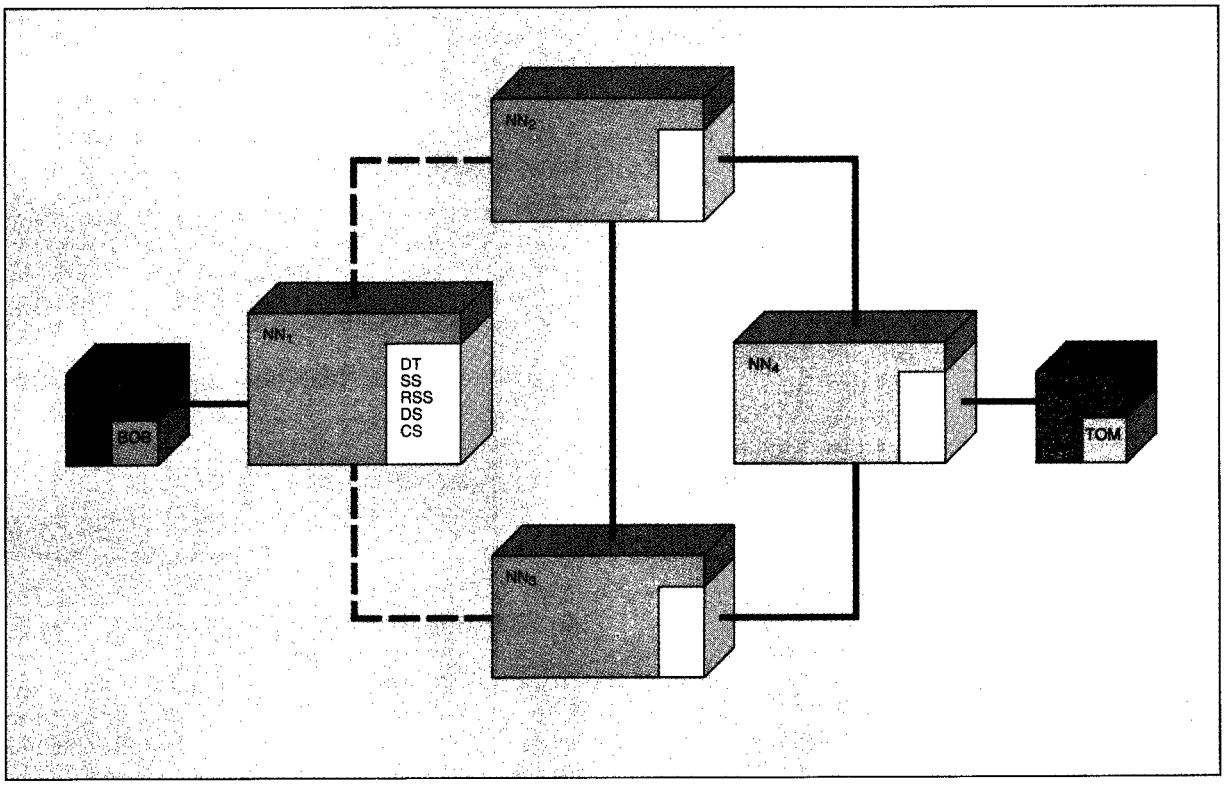
Throughout the APPN work, an attempt was made to understand and address the needs of users of small and intermediate systems, as contrasted with those of the users of large mainframe computers. The technical histories of the many different computer network variants were studied, customer needs were analyzed, new algorithms and protocols for distributed network control were developed, and performance modeling and optimization studies were conducted, all for the purpose of defining the form of networking most suitable for intermediate machines in a low-cost, high-ease-of-use environment. The result was described earlier in an extended technical tutorial.[4]

It is the purpose of this paper to review these motivations for APPN and present a summary of APPN functions, thus setting the stage for the detailed companion paper by Sultan et al. in this same issue.[5]

For purposes of introducing the reader to APPN, it may be described as a network design in which ease of use and simplicity were achieved by (1) decentralizing control so that each node maintains its own responsibility for membership in the network, (2) automating the processes of directory lookup, route finding, session bind, and congestion control in such a way as to make them as invisible as possible to users of the network, and (3) using network control and data transport algorithms that minimize control message overhead and maximize the robustness of the connection between end users.

The clearest exposition of the content of this design can be given by dividing the operation of the network into five phases that take place in chronological order.[4] These are

Figure 1 An APPN network consisting of four network nodes and two end nodes



1. Connectivity services (CS)
2. Directory services (DS)
3. Route selection services (RSS)
4. Session activation (SS)
5. Data transport (DT)

The first four of these are what might be called "transient" or "intermittent" functions, and are collectively referred to as *network control.* They are physically realized by interactions between the *control points* that exist in every network node of the network. The last of the five is the "steady state" phase in which the network does useful work for the users by transferring messages between them. Roughly speaking, we can say that data transport is realized in the lower layers of each node (up through the path control layers of Systems Network Architecture (SNA) and the transport layer in Open Systems Interconnection (OSI) parlance). As shown in Figure 1 (which will be discussed in detail later), there are two parts to an APPN network, one being the set of *network nodes* (NN_1 to NN_4), each capable of intermediate node routing, that collectively make up the *backbone* of the APPN network. The remainder are *end nodes*, e.g., EN_1 and EN_2, that cannot perform intermediate node function, but must have this done for them by a network node. In practice, the end nodes will be those elements of lower communication function such as terminals or print servers, whose number is so large and whose performance and storage capabilities are so modest that there is no point in including them in the backbone. Or they may be processors that are, for operational or organizational reasons, required to be isolated from networking responsibilities. Although there therefore must be at least the two classes—NNs and ENs, it was considered unnecessary and undesirable to add still more classes of nodes and thus to produce a topology having further levels of hierarchy.

Imagine that a user at an end node EN_1 that is attached to a network node NN_1 enters a LOGON request for a session with some application at some other node, and that the name of this target resource

(logical unit, LU) is known but its location, EN_2, is not. In the preparatory *connectivity services* phase, the network node becomes part of the preexisting backbone if it was not already a member. This takes place by activating a data link function with each neighbor (NN_1 to NN_2 and NN_1 to NN_3), and then exchanging enough information between the new node's control point and the other nodes' control points for every node of the newly enlarged backbone to have an updated view of the topology of the backbone (which network node is connected to which). Thus, the addition of a new network node to the network is a dynamic function.

Directory services must now locate the particular network node which either has the target LU within itself or within one of its end nodes. It does so by systematically doing what the protocol literature calls "flooding" of the network with search messages that are propagated outward from the requesting node until the LU is found. Next *route selection services* in NN_1 computes a good route to that target node using an optimum-route-finding algorithm that operates on the topology information that had been created during the connectivity phase. Then the BIND message that had been generated from the original LOGON request flows along that route to the target LU at the target node, and a session between the two is created.

The final steady state *data transfer* phase takes advantage of the fact that when the BIND flowed from source to target (and the bind response RSP(BIND) flowed back along the same physical path), a set of tables had been latched into place in all the intervening nodes so that any subsequent message belonging to that same session between those two end users follows exactly the same path until the session is taken down. During this data transfer phase, special "adaptive pacing" congestion control measures are used that do not throw away messages but also will never deadlock, partly in order that the users will be shielded from having to intervene to recover from a congestion situation.

These are the basic ideas behind the Advanced Peer-to-Peer Networking design. The stimulus to formulate and implement this form of networking grew out of the past history of IBM's intermediate systems, out of requirements being expressed by our customers and others, and from lessons learned from earlier network designs, particularly Systems Network Architecture.[6,7] To set the stage for the accompanying paper by Sultan et al., the remainder of this paper reviews the history of communications in IBM's intermediate systems, indicates how analysis of user requirements and the study of predecessor designs led naturally to APPN, and presents a more detailed discussion of the five phases of operation.

A brief history of IBM intermediate systems communications

In this section we summarize the history of the communication hardware and software for IBM's intermediate systems, beginning with the System/360 Model 20 of the 1960s and culminating with the System/36 of the 1980s. It will be seen that before APPN became available in Release 5 of System/36, intermediate systems could talk to each other only directly over a point-to-point link. Also, the utilization of the available communication support entailed a significant level of communication skills.

Communication hardware of the early processors was quite limited.

Communication hardware of the early processors was quite limited by today's standards. The single binary synchronous communications (BSC) adapter of the System/360 Model 20 supported rates up to 9600 bits/s, as did the multiple adapters of the System/3, but with an optional 56 Kbits/s feature. System/32 had single-line BSC or synchronous data link control (SDLC) support up to 9600 bits/s. System/34 provided either programmable single-line BSC or SDLC adaptors or a four-line adaptor (MLCA—Multiple Line Communication Adapter), both with speeds up to 9600 bits/s, and one of the four MLCA lines could be run at 56 Kbits/s. In addition to BSC and SDLC, one could run X.25 using two MLCA line positions. With the advent of the System/36, the MLCA was improved to provide up to 19200 bits/s per line, again with one line usable to 56 Kbits/s and now with one line per X.25 appearance. The 115 Kbits/s aggregate rate of the new MLCA was raised to 170 Kbits/s with the ELCA (Eight Line Communication Adapter).

As for software, over the last twenty years its function has grown from simple batch BSC to complex interactive SNA. Most of the common link and public data network protocols have been supported: Asynchronous, BSC, SDLC, and X.25 over RS232 and X.21 physical interfaces. The user interface for communications has risen from the assembler subroutine level to native high-level language and has become link-protocol independent.

Significantly powerful communication software began to be introduced with the System/34. Up through the System/32 all communications support had been batch in character. That is, the system acted either as a remote job entry (RJE) terminal to a mainframe computer or supported interactive communications with its subordinate terminals by sending and receiving short batches. The System/34 was a multiple-user system and was the first such system to be cardless; input could be from diskettes or workstation keyboards. The IBM 5250 family of displays and printers provided the workstation for each user. The System/34 supported both BSC and SNA/SDLC protocols, first over analog telephone-grade lines, and then over X.25 and X.21 data networks. Interactive protocols to connected systems were provided by a feature of the System Support Program (the System/36 operating system) called the Interactive Communications Feature (SSP-ICF). SSP-ICF provided three ways of supporting subsystems. The first was program-to-program communication within the same system (that is, using no external communication link). Second, several BSC connections were supported, such as the IBM System/3, 2780 Data Transmission Terminal, 3780 Data Communication Terminal, 3741 Programmable Workstation, 6670 Information Distributor, or another System/34 (as a BSC peer). Communication with Information Management System/Virtual Storage (IMS/VS) and Customer Information Control System/Virtual Storage (CICS/VS) over BSC was also supported. The third type of subsystem supported SNA, e.g., session types ("LU Types") LU 0 to CICS, LU P to IMS, and LU 6.0 between System/34s. A user program became independent of the link protocols since these were handled in each subsystem. Thus, a program written to communicate with CICS transactions using BSC could communicate with the same CICS transaction using SNA without recompilation.

The network topology for all of the System/34 communications was simply point-to-point. Even multipoint networks were logically and physically point-to-point. That is, the control point could have a conversation with any tributary point; any tributary point could have a conversation with the control; but tributary-to-tributary conversations were not supported. These topology limitations of the intermediate systems communications were one of the stimuli underlying the development of APPN.

With the System/36, all of the communication capabilities of the System/34 were retained, and a number of new ones added. SSP-ICF was split into two pieces, one containing the program-to-program subsystems and the other the control logic and user interface. The first retains the name SSP-ICF, whereas the other is called the Communication Feature. This feature merges the logic and user interface functions with the V.25 autocall, X.25, and X.21 support. This packaging allows changes to be made in the two parts independently. For example, some users of Advanced Program-to-Program Communication (APPC—LU 6.2) might need to use the "mapped conversations," which are used for high-level language application-program-to-application-program conversations which do not build the LU 6.2 data stream themselves. Mapped conversations are supported in the APPC Subsystem part of SSP-ICF. The LU 6.2 "basic conversation," which requires the using program (e.g., SNADS and DDM) to build the entire data stream, is included in the Communication Feature. As for other session types, the primary side of LU 0 and 6.2 and the secondary side of LU 0, 1, 2, 3, 6.2, and P reside in the Communication Feature.

There are several important additional higher-level features of System/36 communication support, which we shall now list. For the most part, they use LU 6.2 basic conversations.

Communication and System Management (CandSM) applications on the System/370 host include NCCF/NPDA (Network Communications Control Facility/Network Problem Determination Application), DSX (Distributed Systems Executive), and HCF (Host Command Facility). The System/36 communicates with NCCF/NPDA using the PU-SSCP flow and with DSX and HCF using LU 0 flows. Prior to APPN, the intermediate-level system had to be a PU-Type 2.0 (physical unit) subordinate node connected to a System/370 either directly or through 37X5 communication controllers. With APPN, the small system can be part of a network composed entirely of APPN nodes, with one of them providing access into a Subarea SNA network for purposes of accessing CandSM applications on a System/370.

Personal Services/36 provides SNADS support for document transfer and electronic mail using LU 6.2 basic conversations. The same is true for the 5250 Display Station Pass-Through, which allows a user to log on at one System/36, pass through to another, and then log on to the second system. Once again, the availability of APPN provides these functions through the support of a flexible multihop network topology.

One of the most significant additional System/36 communication features is DDM.

One of the most significant of these additional System/36 communication features is DDM.[3] For present purposes it is sufficient to summarize its function by noting that DDM makes it possible for programs on one system to use standard disk data management routines that normally access local files to access instead files on other System/36s and System/38s and also on System/370s running CICS/OS/VS. Programs running on other systems such as the IBM PC, another System/36, or a System/38 can access data on the System/36. DDM uses the LU 6.2 basic conversation over its sessions.

Before APPN became available, DDM allowed only a primitive, manually controlled, and modest performance form of networking through the Network Resource Directory (NRD) of each DDM instance. A user, knowing that a desired file was at Node C in a daisy chain of nodes A-B-C, could, by sending a request to B and pretending that the file was thought to be at B, count on B's finding that the file was really at C and sending the request to C. With APPN, the request goes directly to C, and does so automatically, without requiring the manual predefinition of the NRDs to tell them about C and the route to reach it.

Analysis of user requirements

By 1982, the IBM experience with intermediate systems and their communication needs had convinced many people that the ability to connect such systems into a network of peers with one another and with larger systems was becoming urgent. In a study done that year, peer networking was identified as one of ten technology areas requiring the most serious attention. Work was started to clarify such a requirement more exactly. The two phases of this process were to make a *user requirements* study and then a *design requirements* study, the latter to act as a definitional bridge between the customer's view of various network functions and the designer's view.

In the user requirements work a task force examined such existing material as GUIDE and SHARE reports; it solicited requirement items from dozens of interested IBM groups and made a priority ordering of the responses; it commissioned three studies by outside consultants; in the end, it developed a coherent picture of the user's needs. From the half-dozen or so different independent sources of input, both inside and outside IBM, the highest priority requirements turned out to be the following interrelated set:

- *Easy to use, change, manage, and grow*—With small and intermediate systems, the system operators and the end users tend to be the same people, usually having neither training nor interest in the communication functions. In such circumstances it is imperative to hide the system details from the user, even at the expense of internal complexity. The problem is made more difficult by the fact that in the small and intermediate business machine environment, changes are much more frequent than in the large mainframe network environment, there being a higher frequency of power on/off cycles, logon/logoff cycles, reconfigurations, moves, and so forth.
- *Peer decentralized network control*—A peer-to-peer dynamic (no coordinated network-wide system definition) style of network control was perceived to be important, because, among other things, it gave to each machine user his or her own control over the machine's membership in the network, without requiring recourse to one or more sources of authority and information. This peer requirement is partly a reflection of the way intermediate and small systems are often used in organizations. Large information processing hosts are usually the responsibility of information centers or headquarters operations, whereas small machines are usually found at the department or subdepartment level. Placing the source of control within the node whose resources are being controlled rather than at some distant node was per-

ceived to offer increased autonomy, speed, and reliability, the last of these by avoiding the "few points of failure" situation.

- *Any topology*—The freedom to configure any topology easily, rather than being restricted to a star, a bus, a hierarchy, or some other pattern, was perceived to offer cost, performance, and ease-of-use advantages.
- *Broad flexibility of physical-level attachments*—Users wanted to be able to make connections easily between nodes using a choice of leased lines, dial-up facilities, and packet-switched services for wide-area connections, and local-area network (LAN) connections for short distances.
- *Interworking with subarea SNA*—Of all the kinds of networks to which a pure APPN network must connect, from the beginning it was perceived that the installed base of SNA networks was by far the most important. With over 15000 SNA networks installed at the time of the requirements work (today over 23000), it was clear that a seamless connection between a pure APPN backbone and a backbone network of SNA subareas was indispensable. Moreover, it was perceived that users would want to use the IBM Token Ring network for physical-level function and LU 6.2 for upper-level function.
- *Simplicity and low cost*—Customers wanted the solution to be easy to understand and, particularly important at the low end, to require very little in the way of main memory requirements or CPU cycles consumed in performing communication services.
- *Continuous operation*—The need for continuous nonstop operation of the network connection was expressed as an emerging requirement, just as it has long been important for networks based entirely on mainframes. It was anticipated that enterprises would increasingly commit their entire business to networks of small and intermediate systems.

This set of requirements was analyzed in early 1983 in a design requirements study which translated these needs into grossly defined features of the design, some of which were to be implemented in all nodes of either the end or network node class (the *base*), and others which were to remain optional feature sets (the *towers*). A decision was made to have the end nodes identical with the PU-Type 2.1 node, which had already been implemented in System/36, System/38, Displaywriter, 5520 Administrative System, and Scanmaster for point-to-point communication, and to design the network nodes to be a set of functions added to the PU-Type 2.1. In this way, customers would not need to change either the hardware or software of their existing end nodes in order for them to be APPN end nodes, and would need to make only modest changes for them to be network nodes.

A task force under D. P. Pozefsky then proceeded with the first phases of defining the protocols and formats required. As this definition work proceeded under the leadership of A. E. Baratz, J. P. Gray, and D. P. Pozefsky, it built particularly on the IBM Research Division results on routing (e.g., Reference 8), distributed algorithms (e.g., Reference 9), and directory function, which had been in progress since 1979.[10] In late 1984, even as APPN was being defined, a joint IBM Yorktown-Rochester effort was begun, led by A. E. Baratz, and this became the System/36 Advanced Peer-to-Peer Networking product.

Some ideas that seemed to work well in earlier networks were incorporated in APPN.

Lessons from the history of computer networks

In designing APPN, some ideas that seemed to work well in earlier networks were incorporated, with suitable changes. In other cases, it was felt that none of the existing designs provided the basis for a proper solution and that new inventions had to be made in order to satisfy the user requirements presented in the preceding section. In this section we highlight a few of the features of preexisting networks that appeared to provide the sort of function that good networks of small systems should have.

ARPANET. Perhaps the most striking feature of ARPANET is the total decentralization of control. In the original ARPANET, nodes (Interface Message Processors) could enter and leave the network at will and a "hot potato" routing algorithm on individual packets would see that they somehow reached the destination, even if sometimes duplicated or out of

order. Later, the routing algorithm was changed to make this process more exact by supplying each node with a "topology database" that captured the node and link connectivity map of the network.[11] In spite of the lack of integrity and performance characteristics sufficient for a commercial environment, the obvious ease of use and availability advantages provided by ARPANET's dynamic reconfigurability and its proven ability to let each node know the entire topology were important things to know when the APPN design began to take shape.

DECNET. Like the original ARPANET, the Digital Network Architecture,[12] which provides the basis of DECNET products, is based on the use of connectionless flows (datagrams) rather than virtual circuits. In order to meet the requirements, APPN avoided the use of a datagram base because of concerns that the performance consequences of end-to-end error and resequencing control would be unacceptable. There was also concern that a network that did not keep track of the physical path by which packets progressed through the network would be one on which it would be difficult to implement completely effective problem determination and other network management function. The routing of successive packets is not constrained, and in fact the method of congestion control used is to throw away overflow packets (an artifice regarded as insufficiently robust for the APPN environment). The layering of the function in DECNET is much cleaner than that in ARPANET, and a good user interface is provided. One of the key features of DECNET has been that communication function is always part of an application processor, so that separate communication front-end machines are not required (e.g., interface processors in ARPANET or communication controllers in subarea SNA). The fact that there is therefore only one hierarchical level (apart from a lower level represented by attached devices) allows a high degree of topological freedom.

Tymnet. Tymnet introduced[13] a particularly efficient method of handling packet routing within an intermediate node. Instead of requiring each node to resolve the address of the final target node into a choice of which outgoing link the packet should go to, Tymnet assigns a number to each virtual circuit on a link and sets in place a simple lookup table in each node that resolves the number into the correct outgoing link (and possibly a new value of the number). The same idea is used in X.25. Tymnet also introduced the idea of controlling packet flow separately within each virtual circuit on each link.

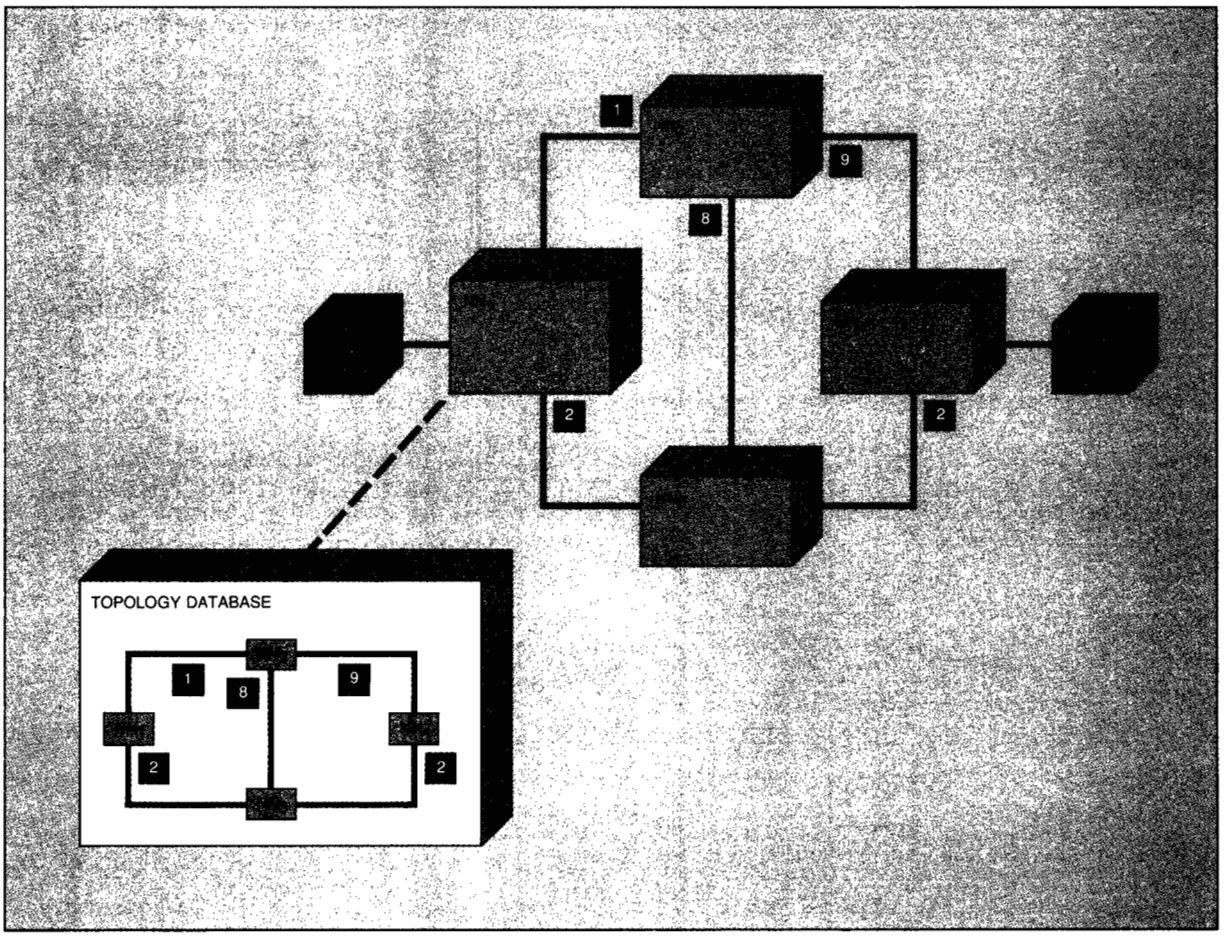
SNA. SNA had the most important influence on the design of APPN, not only because of the strong requirement to be able to interconnect APPN networks to SNA networks, but because certain features of SNA were felt to provide particularly valuable components of the solution. The fact that SNA provides integrity checks at various layer levels (DLC, explicit and virtual routes, sessions and APPC conversations) appeared to serve the availability requirements better than, for example, using connectionless lower-layer flows and providing integrity checking at a higher level (by time-consuming retransmission from the source node). To this end, latching the physical route in place, a feature of SNA explicit routes, is considered indispensable. The availability of Advanced Program-to-Program Communication (session type LU 6.2)[14] seemed to provide, in the form of APPC "conversations," an extremely flexible and robust way of supporting the user's applications and the network's control point-to-control point communication functions. The "negotiable BIND" option in SNA, where both partners engage in a peer dialogue before the session is set up, provides a facility for two LUs to go into session with each other without requiring a master/slave relationship between the two. Finally, a great deal of value was given to the notion of "class of service," in which a source program is able to specify what sort of path properties are needed to best do the work of the session with the target program.

Overview of the APPN design

To provide a tutorial background for the companion to this paper, we now review at an intermediate level of detail the five phases of the operation of APPN listed in the introduction. The discussion here is essentially the same as that given in Reference 4, but somewhat less detailed.

Connectivity services. Consider again the network in Figure 1. In each of the four network nodes and the two end nodes there exists a Control Point (CP), which is responsible for coordinating the first four phases that were listed in the first section of this paper as collectively making up "network control." Imagine that an LU in EN_1 named BOB wants a session with an LU named TOM, whose location (unknown to BOB) is in EN_2, and issues a logon request for a session with him. Also assume for illustration that we are dealing with a dial-up situation in which NN_1 is not yet even connected into the preexisting network consisting of NNs 2, 3, and 4. (The operator of NN_1 has already entered the telephone numbers

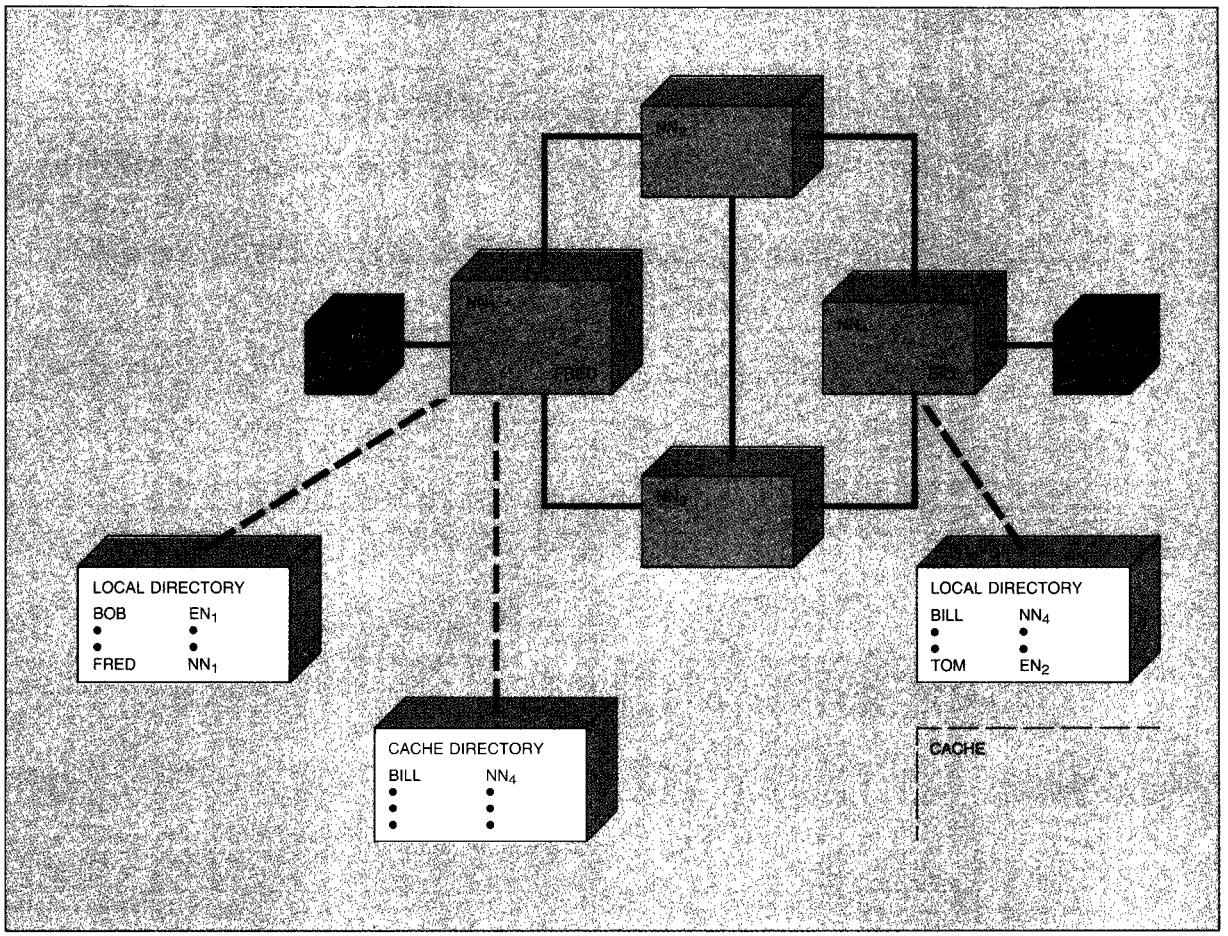
Figure 2 APPN Connectivity Services makes the new node a part of the globally known network topology



required for the two physical connections at node configuration time.) When the control point of NN_1 analyzes the logon request to TOM coming from BOB, it searches for TOM among the local resources and, not finding it, knows it must go to the network to find TOM. NN_1 dials NN_2, and when the physical connection has been completed, a data link level XID (Exchange Identification Information) exchange takes place during which enough information is exchanged for a successful activation of the data link connection to follow. The two control points then go into LU 6.2 sessions with each other by the exchange of BIND and RSP(BIND), whereupon the two CPs exchange information about themselves and synchronize their topology databases. It is at this stage that the new NN_1 learns the topology of the entire network, and NN_2 (and the others) readjust their topology databases to include the new NN_1. This is repeated with the NN_1-to-NN_3 connection. Figure 2 shows the situation at the end of the connectivity phase.

Topology update information flows over the CP-CP sessions in the form of topology database update messages (TDUs), not only when NN_1 enters the network for the first time, but also whenever any significant topology change occurs anywhere in the network, e.g., whenever a link or a node is added or deleted, purposely or by failure. TDUs emanate from the adjacent NNs, fan out throughout the network, and are processed in the nodes through which they pass. As described in Reference 4, the processing algorithms are designed to work without allowing temporary inconsistencies in the topology databases

Figure 3 Directory Services locates the remote target resource



to have any effect, without sensitivity to any chance pattern of failure and then recovery of a link or node, and without requiring safe storage (e.g, on disk). Note that this decentralized way of generating the topology information that each NN will need later for routing eliminates any operator intervention for routing table entry, and in fact allows not only the user, but also network node operators (if any) to be oblivious to the network topology altogether. It also avoids any dependence on one or a few points of failure; if there is any connectivity left in a damaged network, connectivity services will find it and make a "road map" of it.

Directory services. The CP of NN_1 is now in a position to act on the logon request from BOB. As already mentioned, it scans its local directory (Figure 3) to see if BOB is in NN_1 or in any of its subordinate ENs, and then looks to its *cache* directory, its "little black book" of frequently called other parties. If it had scored a hit in the local directory, the search would have been over. Whenever the CP of NN_1 does not score a hit either locally or in the cache, it initiates a distributed search, resulting, in our example, in the discovery that TOM is at NN_4. If it had discovered TOM listed in the cache, it would have performed a directed search, a "probe" of NN_4 specifically, in order to find out if TOM was still there.

The distributed search algorithm does not need to use the topology information. Each NN sends a search message to each of its adjacent network nodes and waits for a reply from each. The algorithm in each node works in such a way that if TOM is anywhere in the network, he will be found, in spite of arbitrary patterns of link and node failure and recov-

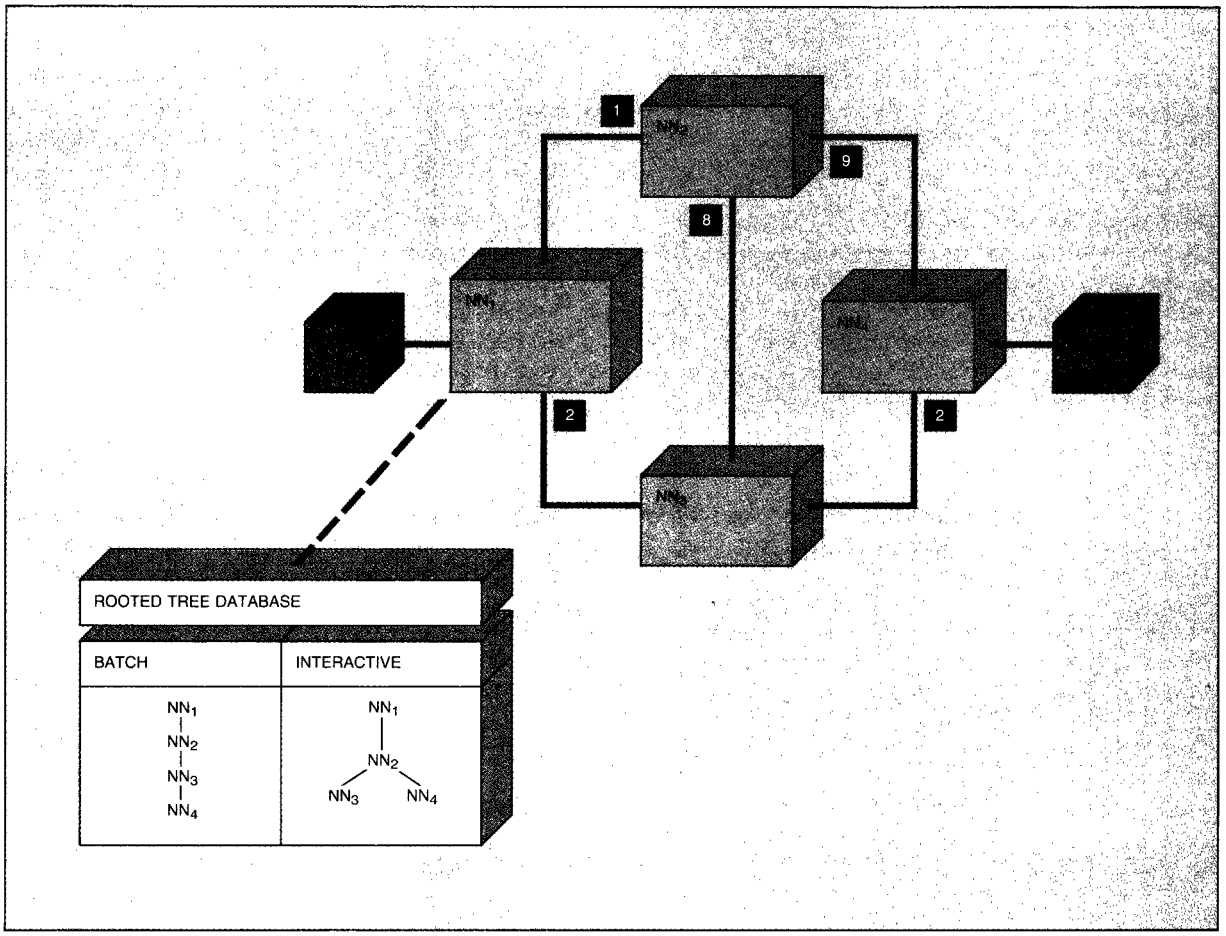
ery, and with an overhead of no more than two messages per link.[9]

Route selection services. Once it has been determined that TOM is at NN_4, the topology database can be used, together with information provided by BOB concerning what properties of the route are important to him, to determine the preferred route through the network from NN_1 to NN_4. Which physical route to choose depends on which "class of service" (COS) was specified indirectly in BOB's original BIND message to NN_1. If the session is to be an interactive one, the route must have the fastest possible response time, even if at the expense of some bandwidth, whereas for batch file transfers or remote job entry the converse is true, and perhaps a different physical route is better. And there are other examples, such as a "security" COS which dictates use only of encrypted links on all hops making up the route.

Route finding consists of two stages (in addition to the preceding topology database part of connectivity services). The first phase is the preparation of a *rooted tree database* (Figure 4) which captures in each NN all the preferred paths to each other NN, one rooted tree for each COS, as shown in the figure. The rooted tree is computed in every network node every time a change in topology and a subsequent LOGON request make it necessary. The calculation uses the link characteristics as broadcast in the topology database update messages.

Each rooted tree database is created by running any one of several well-known shortest-path algorithms

Figure 4 Route Selection Services provides good paths to each network node



(see Reference 4), where "shortest" does not necessarily mean "physically shortest" but having the smallest algebraic sum of link properties of the links (hops) making up the route. For example, for the interactive COS, each link property would simply be its delay (perhaps with a small weight given to cost to prevent selecting excessively high-speed and expensive lines).

The second phase consists of simply accessing the rooted tree database and building a Route Selection Control Vector (RSCV) to be appended as part of the BIND so that the BIND will thread its way to the right destination over the right route, in this case NN_4. The RSCV is simply an ordered list of the nodes and links to be traversed, in the case of our example

NN_1 - LK_1 - NN_2 - LK_8 - NN_3 - LK_2 - NN_4

Note again that no function in a remote (e.g., centralized) control node needs to be accessed and that no operator functions, such as loading of routing tables, are required; the process is unseen by the user and any operator.

Session bind. After the route selection control vector has been appended to the BIND, the latter is sent from NN_1 to NN_4 (via NNs 2 and 3). This causes two things to happen.

First, when the BIND arrives at the LU called TOM, a negotiation of properties the session is to have takes place, and if an agreement is reached, TOM sends back to BOB an RSP(BIND) message. Note that this is accomplished entirely between BOB and TOM, without recourse to some central control point to preside over the process.

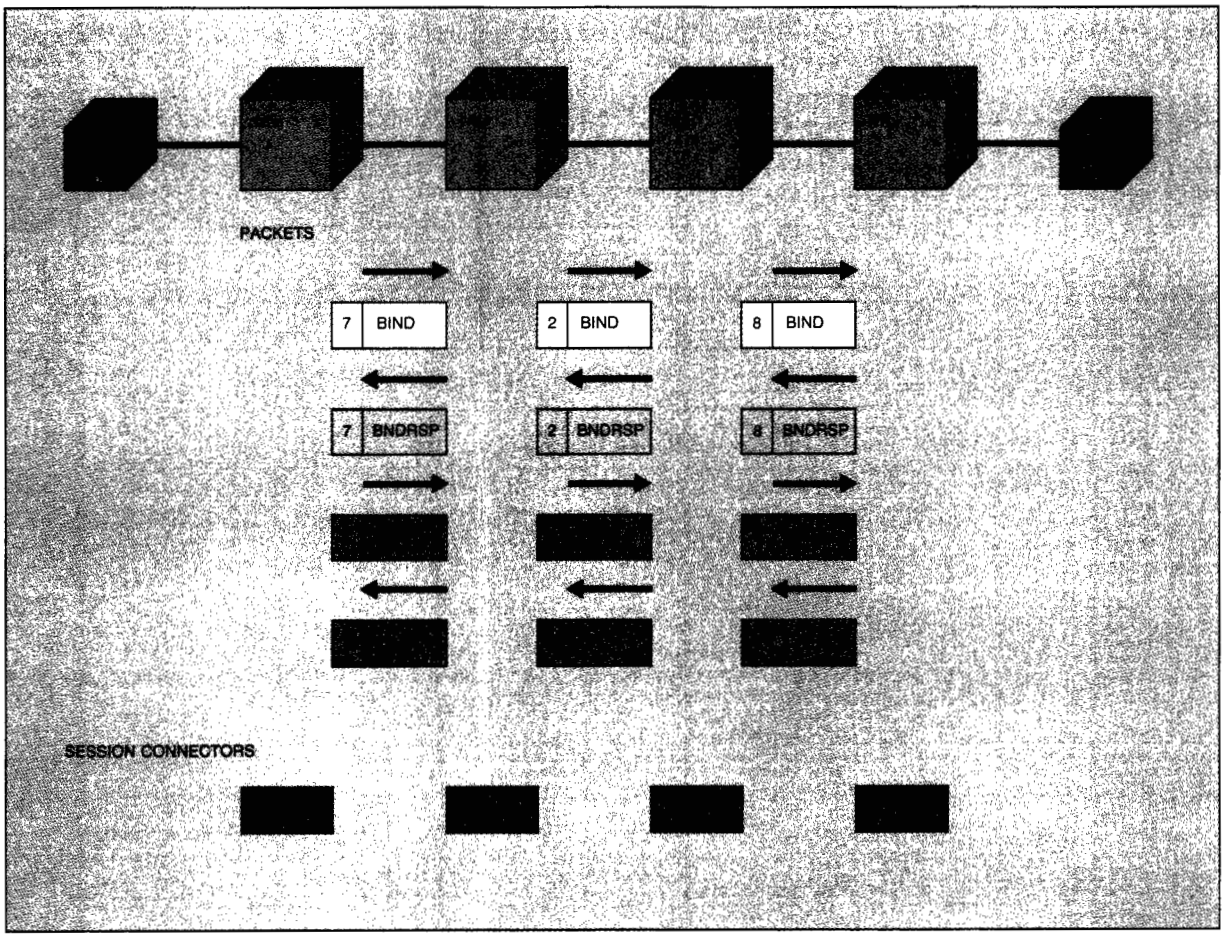
Second, as the BIND made its way along the path, a set of Session Connectors was put in place, as shown in Figure 5. These are swap tables that map a 17-bit number ("session identifier") in the transmission header (TH) of the incoming message into a similar number in the TH of the outgoing message. The incoming value to NN_2 of the session identifier was picked at NN_1 as one that was not already in use on that particular Link 1 (although it might have been in use on another). As the BIND passes through NN_2, the CP of NN_2 picks a 17-bit value not already in use on its Link 8 to NN_3 and builds the session connector, and so forth for the other NNs farther down the path. Once the session connectors have been latched in place along the route, every succeeding packet in that session is made to follow the same route by simply giving it the right 17-bit session identifier at NN_1 and sending it out the correct Link 1. Note that by using swappable session indicators instead of network addresses one may have 17 bits worth of addressability *per link* rather than *per network*, as well as the benefit of reduced processing load that the short header field and simple table lookup of session connectors incurs. (Even though the session indicator is therefore not an address in the usual network-wide sense, it will occasionally be called by that name in the paper accompanying this one,[5] since the particular FID-2 transmission header field used for it originally served the function of addressing spatially distinct entities.)

The APPN flow control is aimed specifically at providing fairness and guaranteed freedom from deadlock.

Data transport. Once the session has been set up between LUs such as TOM and BOB, there are several requirements on the flow of packets within the session, of which the most important are efficiency, fairness, and freedom from deadlock. Efficiency means that the packets flowing in the session must be delivered to their destination as rapidly as possible by neither underutilizing the communication resources, particularly buffers in the NNs, nor overdriving them so that packets are lost and recovery actions are required. Fairness means that if a bottleneck is caused by one particular session within a class, the remedial measure of throttling back packet flow is exerted against that session. Deadlock, a condition that reduces the availability as well as the ease of use of the network, is the situation where, for some reason, a node cannot rid itself of an outgoing packet until it receives on one of its links some other packet, but this other packet will never be sent until the first one is sent. An operator intervention is required in most networks to undo this situation.

Two measures are used in APPN data transport to deal with these problems. Segmenting/reassembly is the breaking up of RUs (Request/Response Units

Figure 5 Session connectors and the flow of packets



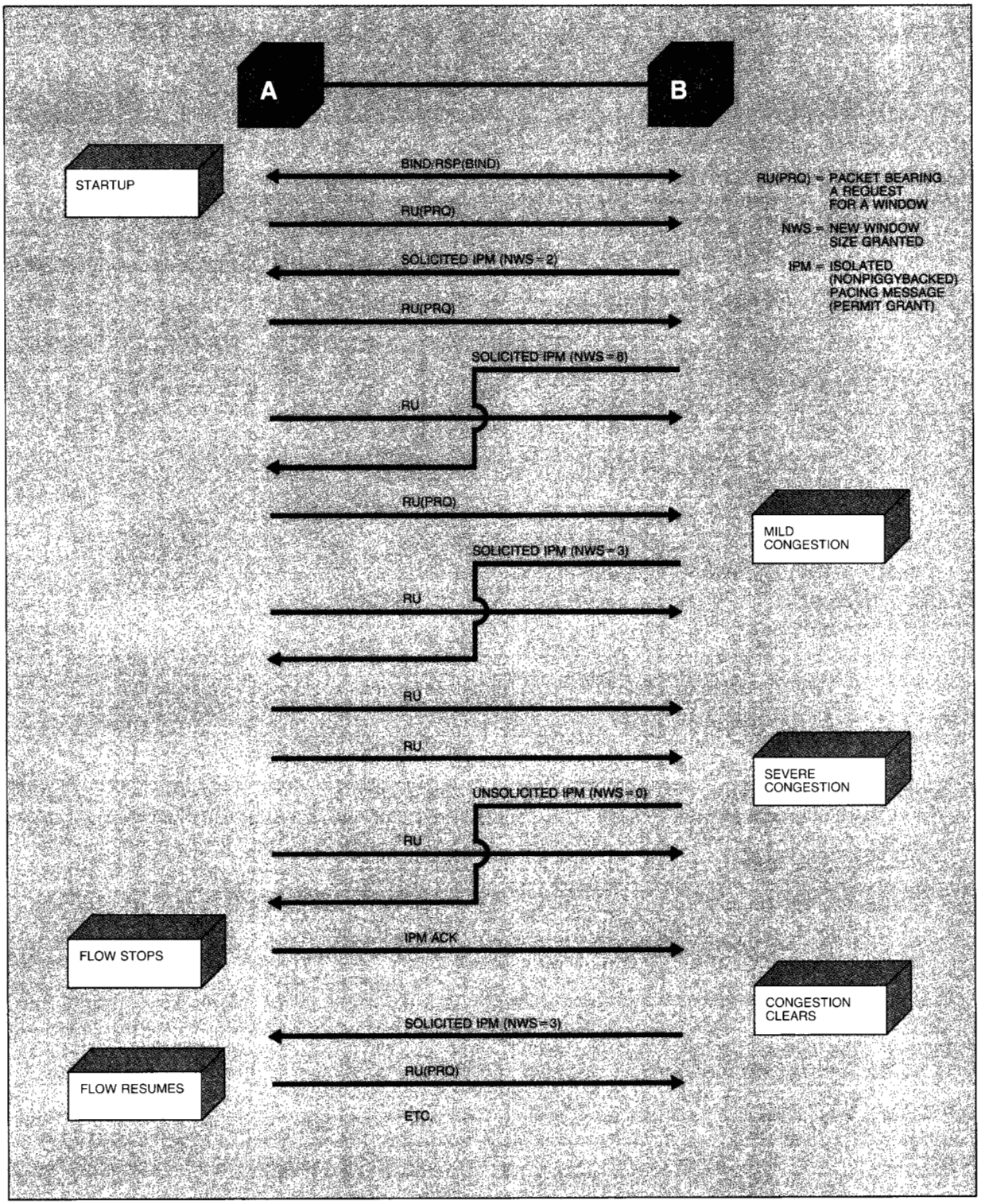
from the higher layers) into smaller message segments so as to accommodate the buffer size of the next node on the route, and then putting the RU back together again at the final node. In APPN, segmenting can take place essentially only at the first network node on the route. The second, flow control, required considerable new thought.

The APPN flow control is aimed specifically at providing fairness and guaranteed freedom from deadlock. Although it is also efficient, it is not quite as efficient as would have been some alternatives that allowed rare but possible deadlocks. The requirement for freedom from deadlock was considered unavoidable in a network of small processors where the operator is often the unskilled user. Instead of controlling the flow on each session separately end to end, or each hop separately for all sessions collectively, APPN controls flow on each hop of each session. This maximizes fairness by throttling back on that session and that hop within a session that caused the problem.

Efficiency is maximized by basing the flow control on being able to vary the size of a "window," a number of RUs (not the number of message segments) that a sender on a link (hop) is allowed by the receiver to send before asking for another window of RUs. The procedure is illustrated in Figure 6.

Summary. Although it will be found on detailed examination that APPN uses, and extends in many cases, well-known algorithms or techniques, it has proved not at all simple to adapt these ideas to the small-system, high-ease-of-use environment. The accompanying paper[5] shows how redesign and extrap-

Figure 6 Sample APPN flow control scenario



olation of existing ideas and the invention of new ones allowed these objectives to be achieved. APPN is a good example of the use of internal sophistication and complexity to achieve compactness and external simplicity.

Concluding remarks

This paper has attempted to set the stage for the paper by Sultan et al.[5] which will discuss the experience gained in the System/36 implementation of Advanced Peer-to-Peer Networking.

We have intended to portray in this paper the fact that APPN was a synthesis arising from three sources: (1) a study of expressed needs of customers, particularly those having the most sensitivity to what the technological future might offer, (2) the state of the computer network art, together with an identification of the "choke points" at which further technical innovation was required (especially in distributed algorithms) in order to achieve the desired properties, and (3) the whole twenty years of experience with intermediate systems and the needs of their developers for expanded communication function.

We consider APPN to hold considerable promise for a future involving great improvements in transmission technology, greater reliance of businesses on nonstop network operation, and a broadening of that portion of the work force that will be able to use IBM's networks with ease and assurance.

Acknowledgments

The authors would like to express their esteem and gratitude to a number of people who played particularly important roles in the process of identifying the small-system networking requirements and managing the APPN solution to them. They include especially Lewis M. Branscomb, Barry C. Goldstein, Peter C. Huffman, Jeffrey M. Jaffe, Jean Lorrain, F. H. Moss, Diane P. Pozefsky, Mark Pozefsky, Larry K. Plank, and Robert J. Sundstrom. We thank Mark Pozefsky particularly for help in improving the present paper. Names of other people who also contributed to the management and requirements parts of the effort are to be found among the authors and persons acknowledged in the paper by Sultan et al.[5]

Cited references and note

1. Advanced Peer-to-Peer Networking was announced as part of Release 5 of System/36 in June 1986.
2. B. Housel and C. J. Scopinich, "SNA Distribution Services," *IBM Systems Journal* **22,** No. 4, 319–343 (1983).
3. *IBM Distributed Data Management Architecture, General Information*, GC21-9527, and *IBM Distributed Data Management Architecture, Reference*, SC21-9526, IBM Corporation; available through IBM branch offices.
4. A. E. Baratz, J. P. Gray, P. E. Green, J. M. Jaffe, and D. P. Pozefsky, "SNA networks of small systems," *IEEE Journal on Selected Areas in Communications*, 416–426 (May 1985).
5. R. A. Sultan, P. Kermani, G. A. Grover, T. P. Barzilai, and A. E. Baratz, "Implementing System/36 Advanced Peer-to-Peer Networking," *IBM Systems Journal* **26,** No. 4, 429–452 (1987, this issue)
6. J. P. Gray and T. B. McNeill, "SNA multiple-system networking," *IBM Systems Journal* **18,** No. 2, 263–297 (1979).
7. R. J. Sundstrom, J. B. Staton III, G. D. Schultz, M. L. Hess, G. A. Deaton, L. J. Cole, and R. M. Amy, "SNA: Current requirements and direction," *IBM Systems Journal* **26,** No. 1, 13–36 (1987).
8. J. M. Jaffe, F. H. Moss, and R. A. Weingarten, "SNA routing: Past, present, and possible future," *IBM Systems Journal* **22,** No. 4, 417–434 (1983).
9. J. M. Jaffe, A. E. Baratz, and A. Segall, "Design issues in the implementation of distributed, dynamic routing algorithms," *Computer Networks and ISDN Systems* **12,** No. 3, 147–158 (1986).
10. P. E. Green, *Prospects for Peer Decentralized SNA Networks*, Research Report RC-7526, IBM Corporation, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (February 6, 1979).
11. J. M. McQuillan et al., "The new routing algorithm for the ARPANET," *IEEE Transactions on Communications* **COM-28,** No. 5, 711–719 (May 1980).
12. S. Wecker, "DNA, the Digital network architecture," Chapter 10 of *Network Architectures and Protocols* (edited by P. E. Green, Jr.), Plenum Publishers, New York (1982).
13. L. W. Tymes, "Routing and flow control in TYMNET," *IEEE Transactions on Communications* **COM-29**, No. 4, 392–398 (April 1981).
14. J. P. Gray, P. Homan, P. J. Hansen, M. A. Lerner, and M. Pozefsky, "Advanced Program-to-Program Communication in SNA," *IBM Systems Journal* **22,** No. 4, 298–318 (1983).

Paul E. Green, Jr. *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Green is a Research Staff Member in the Computer Science Department. He received the Sc.D. degree from the Massachusetts Institute of Technology in 1953. After many years at M.I.T. Lincoln Laboratory, he joined IBM Research in 1969 as Senior Manager. His interests have centered on signal processing and computer network architecture. Dr. Green has had several tours of duty on the IBM Corporate Technical Committee and the Engineering, Programming, and Technology Staff at Armonk, New York. He recently received an IBM Outstanding Innovation Award for his work in formulating and promoting Low Entry Networking. Dr. Green is a Fellow of the IEEE and a member of the National Academy of Engineering.

Robert J. Chappuis *IBM System Products Division, Highway 52 and Northwest 37th Street, Rochester, Minnesota 55901.* Mr. Chappuis joined IBM in 1958. Since that time he has held numerous technical and managerial positions dealing with SNA architecture and implementation. He participated in the corporate task force to define the converged SNA logical unit, LU 6.2. In

1982, he was a member of the SPD technical staff responsible for the initiation of the APPN strategy and direction. He is currently the Program Manager of Communications Software Development at the Rochester Programming Center.

Jan David Fisher *IBM System Products Division, Highway 52 and Northwest 37th Street, Rochester, Minnesota 55901.* Mr. Fisher joined the Data Processing Division of IBM in Wichita, Kansas at the Wichita Branch Office in 1965 as a Systems Engineer. In 1974 he joined the General Systems Division in Rochester as a Process Industry Technical Marketing Support Representative. He spent seven years as a product planner for the System/34 and System/36 communications software support. Mr. Fisher is currently a Senior Architect-Planner for Distributed Data Management Architecture.

Paul S. Frosch *ROLM Corporation, 4678 Alpine Meadow Lane, Colorado Springs, Colorado 80919.* Mr. Frosch joined IBM in June 1966 at the East Fishkill, New York, laboratory, working at first with development pilot lines in setting up systems for analyzing process/product data in order to increase yields. He subsequently was named database administrator for an IMS/VS DB/DC system implemented for East Fishkill manufacturing. In 1973, he transferred to the former Data Processing Division in Rochester, New York, as a systems engineer. He worked with the Xerox account on the design and implementation of a large-scale IMS DB/DC system for their manufacturing and sales organizations. In 1977, he joined the marketing support systems organization (DPMG) in Harrison, New York. He held several staff and management positions, focusing on DB/DC productivity solutions for large and intermediate systems users. In 1982, Mr. Frosch joined the ISG telecommunications product line evaluation department in Rye Brook, New York, concentrating on requirements for SNA software and communication and systems management products. He led strategic requirement task forces on small-systems communications (LEN/APPN) and worldwide Open Systems Interconnection (OSI). In 1985, he joined the teleprocessing products organization in IS&CG as manager of TPO product programs. He became manager of Redwood product marketing upon joining ROLM in Colorado Springs in 1986, where he is currently responsible for the product management of the Redwood voice/data controller. Mr. Frosch received his B.S. in mathematics from the City College of New York in 1966 and graduated from IBM's Systems Research Institute in 1969. He has done graduate studies in computer science.

Chuck E. Wood *IBM Communication Products Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Mr. Wood received his B.S. in mathematics and business administration from Mankato State University in 1968. He joined IBM in 1969, working in an information systems group. In 1971 he joined the product development group responsible for the microcode for the IBM 3741/2 Data Station. From 1975 until 1982 he helped test and design, and then managed, various SNA communication facilities for the System/32, System/34, and System/36. In the fall of 1982 he moved to White Plains, where he worked on the communication strategy for the System Products Division. He transferred to Research Triangle Park in October 1984, managing a design group for the IBM Token-Ring Network. In 1986 Mr. Wood joined the Communications Programming Laboratory, where he currently works in the advanced design technical staff organization.

Reprint Order No. G321-5305.