SNA Perspective

Volume 14, Number 3
March, 1993
ISSN 0270-7284

The single source, objective monthly newsletter covering IBM's Systems Network Architecture

APPC Goes Full Duplex

Until recently, Advanced Program-to-Program Communication (APPC) conversation flows have been limited to two-way alternating (half-duplex) protocols. With half-duplex protocols, only one program has permission to send at any point in the conversation. Permission to send alternates between a program and its partner under program control. Recently, APPC has been enhanced to also allow two-way simultaneous (full-duplex) flows on a single conversation. With full-duplex protocols, both programs always have permission to send.

Half and *full* are not value terms, however, with *full* implying more or better. Though some applications will be much better served by full-duplex conversations, many applications will continue to work better with half-duplex conversations. This article will:

- Help you understand this new full-duplex feature
- Show you how to decide when to use this feature in your application
- Describe how an application would use this feature through enhanced calls proposed for Common Programming Interface for Communications (CPI-C) Level 2.

(continued on page 2)

APPN+ Takes Shape: A Preview of High Performance Routing

Advanced Peer-to-Peer Networking (APPN), which first appeared in products in 1987, has been evolving steadily. In 1991, IBM began publicly referring to a significant change in APPN which it informally called APPN+. More recently, IBM has begun to discuss some details of the protocols that are at the heart of this change. Although these protocols will not be seen in products for at least a year, understanding APPN's direction is important to network planning today.

This article introduces these new protocols, which are together called High Performance Routing (HPR). We discuss how HPR differs from and relates to APPN's current Intermediate Session Routing (ISR). We describe the four leading types of routing protocols—virtual/explicit routing, label swapping, destination routing, and source routing—and discuss how they relate to ISR and HPR. We also note nine concerns about APPN and examine which ones HPR addresses.

(continued on page 12)

In This Issue:

APPC Goes Full Duplex 1
Finally, APPC conversations can use full duplex or half duplex. We list applications that would most benefit from full duplex as well as the advantages of half duplex. No special configuration is needed to add full duplex and the same session can be used for both types of conversation.

APPN+ Takes Shape: A Preview of High Performance Routing 1
HPR promises higher performance and better congestion control than APPN's current ISR as well as rerouting around failures. Drop-in migration is supported: HPR nodes and ISR nodes can coexist. HPR is not a panacea—we discuss several remaining concerns about APPN. But its benefits are welcome.

Architect's Corner
Our architect is on vacation.

(continued from page 1)

From the beginning, APPC was designed to reduce and simplify the programming effort required to perform effective program-to-program communication.

- **Allocate.** Instead of requiring the programmer to code logic that identifies and separates different requests from a single stream or logic that correlates replies back to their requests, APPC provides an Allocate call. Client programs use Allocate to obtain a conversation with a server program and APPC automatically activates the necessary network resources, starts an instance of the server program, and connects the server program to the conversation as necessary. To identify and separate different requests, each is given a different conversation. To correlate replies back to their requests, replies are returned on the same conversation that carried the request.
- **Send_Error.** Instead of requiring the programmer to code logic to purge inbound records after discovering an error that makes it impossible to process them, APPC provides a Send_Error call. Send_Error informs the partner program of the error and automatically purges any records that may have been sent before it found out about the error.
- **Confirm.** Instead of requiring the programmer to code logic that verifies that the partner program has received and successfully processed all the data sent so far, APPC provides a Confirm call. Confirm suspends the conversation, informs the partner program that confirmation has been requested, and notifies the caller of the partner's response to the request. The partner either responds with a Confirmed call when all data is processed successfully or responds with a Send_Error when an error is detected while processing the data. Just as it does for many other race conditions, APPC automatically handles the cases where an error is detected before the confirmation notification arrives or before confirmation is requested.
- **Resource Recovery.** Instead of requiring the programmer to code logic that keeps audit trails and negotiates recovery procedures when the network fails in the middle of a database update, APPC works together with the Resource Recovery system to automatically commit or backout the update when the network is restored.
- **Deallocate.** Instead of requiring the programmer to code logic that informs the partner program that the conversation is over, APPC provides a Deallocate call that supports both conditional and unconditional termination. Conditional termination includes a confirmation request and the conversation is terminated only if the partner processes the data successfully and responds with a Confirmed call. If the partner detects an error and responds with Send_Error, the conversation continues so the programs can attempt to recover.
- **Buffering.** Instead of requiring the programmer to code logic that builds application records to utilize the network's transmission frame effectively, APPC automatically provides a buffering service. APPC minimizes the network overhead by packing as much data as possible into a network frame and sending the frame only when it overflows, when the program uses the Flush call (because a delay between Send_Data calls is expected), or when the program is done sending (makes a Receive call to receive data, requests a confirmation, or deallocates the conversation).

These powerful features are possible only because a half-duplex conversation enables APPC to make the following assumptions about the relationship of the records being sent and received.

- The next record sent depends on the successful processing of the last record sent.
- The next record received depends on the successful processing of the last record received.
- The records to be sent by the server depend on the records last sent by the client.
- The records to be sent by the client depend on the records last sent by the server.
- From the beginning of the transaction to the end, the program uses the conversation for only one unit of work at a time.

For many applications, half-duplex conversations simplify the application logic and enable the application to use the many powerful features provided by APPC. For example, in a query transaction, the client sends a request to the server and then waits for the server to send a reply. Likewise, in a file transfer transaction, the client repeatedly sends records to the server and then waits for the server to confirm that the file was received and stored successfully.

Some more complex application designs require the programs to send and receive data simultaneously. APPC was enhanced to allow full-duplex conversations in order to support these applications that cannot tolerate the half-duplex flow restrictions. Even though these applications would not use many features provided by APPC, the full-duplex enhancement provides a common open application interface that is easily integrated into existing and future networks.

To understand which application designs benefit from full-duplex conversations, consider the following application models that need to send and receive data simultaneously.

Full-Duplex Application Models

The three full-duplex application models are:

- Protocol transport applications
- Transaction pipeline applications
- Real-time control applications

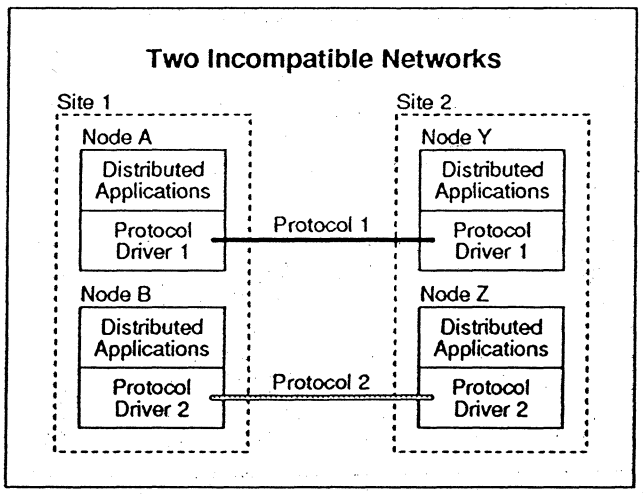

Figure 1

These models and their characteristics are described below.

Protocol Transport Applications

A protocol transport application enables customers with incompatible networking protocols between two sites to share network media by using a preferred protocol to transport an incompatible protocol.

For example, because the networking protocols in Figure 1 are incompatible, they cannot share a common physical medium. The customer must pay for two physical connections between the same two sites.

In most cases, additional investment in data communication hardware and software can be justified if the configuration can be simplified to a single physical connection. One such investment would be a protocol transport application, as shown in Figure 2.

Protocol transport is a full-duplex application. As far as the transport applications are concerned, the records sent are completely independent of the records received. Above the application, a complete protocol machine and, optionally, other protocol stacks above it coordinate the end-to-end protocols.

The example in Figure 3 (see page 4) illustrates how an APPC full-duplex conversation can transport a protocol such as X.25 or NetBIOS. The protocol transport application adds an envelope to a frame and sends it within a conversation record. On the receiving side, the application removes the envelope

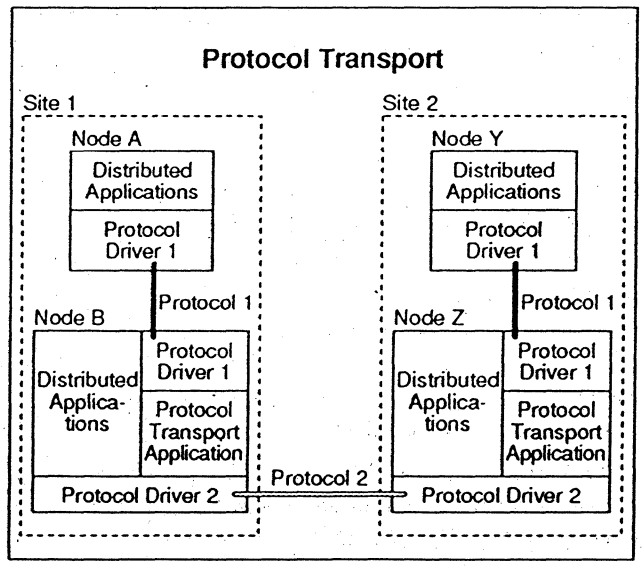

Figure 2

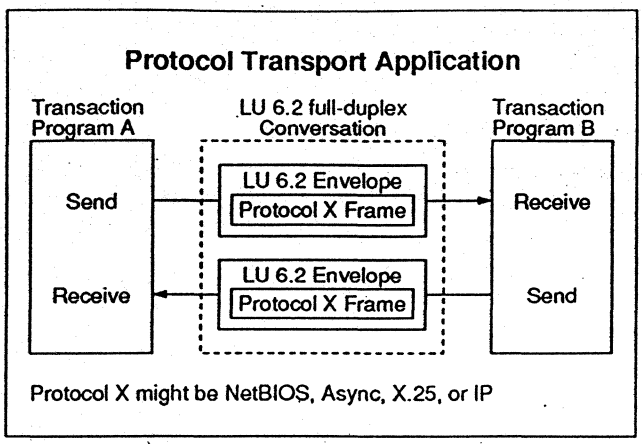

Figure 3

before delivering the frame to the target drivers. Although this example shows how data link control frames can be transported through an SNA network, the same design could be used for higher level protocols. For example, a protocol transport application could transport TCP streams.

Transaction Pipeline Applications

Transaction pipeline applications interleave requests and replies from many different application subfunctions onto a single full-duplex conversation. The requests are generally unrelated messages.

Consider an in-store controller with a transaction rate of 25 credit check transactions per second and an average transaction response time of one second. On average, then, 25 different transactions are active at the same instant. If each transaction runs with a half-duplex protocol, the controller needs 25 simultaneous conversations and 25 copies of the network resources (sessions, buffer pools, and so on) to support those conversations. This memory and execution overhead is a heavy load for a small controller.

Transaction pipeline applications reduce the network load in this type of environment. Rather than using a half-duplex conversation to correlate the records exchanged to perform a transaction, the application implements its own *explicit correlation* function. When a new transaction is started, the application assigns a correlation value (often a control block pointer) and places the value into all records exchanged to perform that transaction. When a record arrives, its target transaction can be determined by the correlation value it carries.

By using the correlation values, the applications can place all the traffic onto a single conversation with no ambiguity. No matter how many transactions run simultaneously, all the records to be sent can be funneled into a single conversation because the partner can use the correlation values to fan them out again.

For the conversation, the records sent and received are completely independent. The relationship between the records is managed by the correlation protocol machines implemented in the application. These correlation protocol machines determine all the end-to-end protocols.

The scenario illustrated in Figure 4 interleaves requests and responses from several different subfunctions (named X, Y, and Z) onto a single full-duplex conversation. Both transaction programs maintain correlation tables. The individual requests are unrelated and considered to be different work units.

It is possible for a transaction pipeline application to be designed using half-duplex conversations. Instead of using a single conversation for all requests and replies, the programs would use many short conversations, one for each message. Each message sent includes Allocate, Send_Data, and Deallocate type(FLUSH).

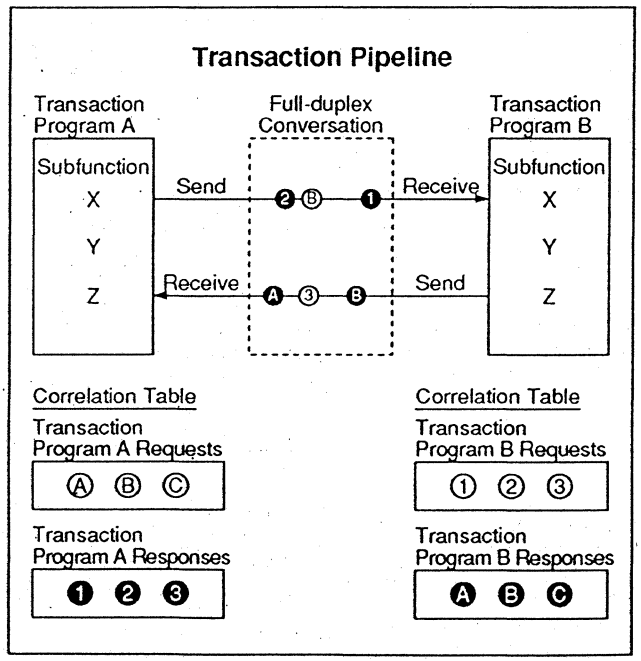

Figure 4

If granular security and accounting are required, the half-duplex design has the advantage of a user ID automatically associated with each message. If parallel processing and horizontal growth is required, the half-duplex design has an advantage because each message has its own instance of the server, the servers run in parallel and, optionally, the servers run on more than one computer. However, if the network fails and a message is lost, the program will not know unless it uses time-outs or some other technique for sensing lost messages. Furthermore, the order in which the messages arrive may be different from the order in which they were sent.

The full-duplex design described above, on the other hand, uses just one conversation for all transactions and therefore does not have the overhead of allocating conversations and starting servers for each transaction. If the network fails, both programs using the conversation are notified and the recovery of lost messages can begin immediately. Since the same conversation is used for all messages, messages are always sent and received in the same order.

However, there are disadvantages. APPC's automatic security and accounting will be performed only when the conversation is started, not for each of the transactions carried by the conversation. Furthermore, all transactions go through a single pair of programs. For parallel processing and horizontal growth, the logic must be added to the application instead of using the services provided by APPC.

Real-Time Control Applications

A real-time control application simultaneously sends command messages and receives feedback messages. The application uses information from the feedback messages when generating new command messages, but there is no relationship between the number of messages sent and received. Applications can combine this model with the transaction pipeline application model and interleave many commands and feedback messages on a single full-duplex conversation.

Imagine a manufacturing plant where a computer is directing the actions of automated systems on the plant floor. The messages flowing out of the computer control the equipment: turn up the heat on oven number 4, start conveyer A. The messages flowing back to the computer provide feedback: the oven temperature is now 1000 degrees, conveyer A is jammed, sensor Z has tripped.

In some cases, the feedback has real-time requirements. If the computer does not respond within the required time, a disaster may occur: an oven may explode or people may be injured. The feedback cannot be delayed while waiting for the partner application to grant permission to send or while waiting for a new conversation to be allocated.

In other cases, the feedback is loosely related to the commands being sent. The computer performs real-time trend analysis to determine what action to take next. There is no fixed relationship between the number of feedback messages and the number of command messages. In other words, if the application attempts to use half-duplex protocols, there is no way to determine which program should have permission to send.

An example of a real-time control application appears in Figure 5. The furnace sends temperature readings to the controlling device. In response to certain temperature thresholds, the trend analysis function in the controller sends temperature adjustment commands to the furnace.

A real-time control application is another application that lends itself to full duplex. The conversation is unaware of any relationship between the records

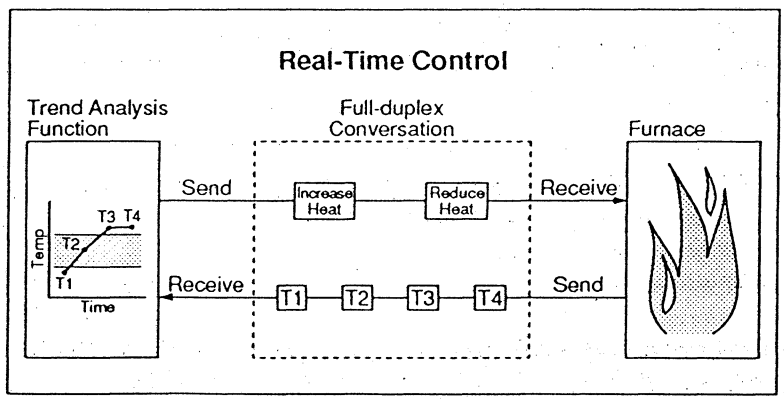

Figure 5

How can CPI-C Level 2 support full-duplex conversations on APPC platforms that do not implement the full-duplex conversation option set?

When APPC supported only half-duplex conversations, programmers invented clever ways to implement full-duplex applications using two half-duplex conversations. With this technique, the client program allocates a conversation to the server program and then the server allocates a conversation back to the client. The client's conversation is only used to send data from the client to the server and the server's conversation is only used to send data from the server to the client. CPI-C Level 2 will use this technique to emulate full-duplex conversations on APPC platforms that do not implement the option set for full-duplex conversations.

exchanged. Instead, the application contains a feedback protocol machine that understands the relationship between the records.

A similar real-time control application is designed with half-duplex conversations using one of the two APPC features that can interrupt a program that is sending: Send_Error and Request_To_Send. But if a program uses Send_Error to interrupt its partner, obtain permission to send, and send a feedback message, the side effects may be unacceptable. First, Send Error will suspend the caller until the partner program makes a call that reports the condition. Second, any data sent by the partner but not received will be purged and must be resent by the partner after the feedback message is received.

Instead, if a program uses Request_To_Send to ask the partner for permission to send, no data will be purged. However, the program is unable to send the feedback message until three conditions are met—the program has received the data that has already been sent by the partner, the partner makes a call that returns the Request_To_Send indicator, and the partner grants permission to send.

Using full-duplex design allows the program to send a feedback message without delay and without purging valid data, assuming that the partner program maintains an active Receive. This is a definite advantage when it is critical that the feedback message be processed as quickly as possible.

Contrasts: Full-Duplex and Half-Duplex Applications

Related Records

Full Duplex. With full-duplex applications, only the application knows how exchanged records are related.

- Records from different application subfunctions are interleaved on the same conversation.
- Some subfunctions receive records while other subfunctions send records.
- If the conversation calls are used to synchronize operations or purge records made obsolete by an error detected by the application, it will inadvertently suspend subfunctions that don't need to be suspended or purge records that are still valid.
- Every full-duplex application has some type of protocol machine to manage the relationship between records. Examples include a communication protocol stack for protocol transport applications, a request correlator for transaction pipeline applications, and a feedback analyzer for real-time control applications.

Half Duplex. With half-duplex applications, the application and the conversation understand how exchanged records are related.

- The application exchanges a set of related records and uses a conversation for only one unit of work at a time.
- The application never sends and receives data at the same time.
- Since the records are all related, a conversation call can safely suspend the caller when synchronizing programs and can easily identify and purge records made obsolete by an error detected by the application.

- Every half-duplex application uses the conversation to manage the records for a given unit of work. It does not need the extra logic of a protocol machine to perform its task.

Given that full-duplex applications require extra logic where half-duplex applications simply utilize services provided by the conversation, why would a programmer choose to implement a full-duplex application and take on this extra programming burden? From the above models, you can see the primary trade-off: network resource utilization versus application complexity.

Fundamental Primitives

Whether you use half-duplex or full-duplex protocols, program-to-program communication has several fundamental primitives:

- Identifying a transaction service
- Starting a conversation between a client and a server, automatically starting the server if necessary
- Stopping a conversation
- Sending data related to the transaction
- Receiving data related to the transaction
- Forcing data to be transmitted
- Reporting errors and purging data that is invalidated by the error
- Requesting confirmation that the data sent has been successfully processed (in other words, synchronizing the programs)
- Granting or rejecting confirmation requests

Half Duplex. With half-duplex protocols, the APPC communication subsystem can easily provide the formats and protocols needed to implement these primitives on behalf of the application. The communication subsystem encodes the necessary information, manages the protocol state, and resolves the race conditions. The application manages the application protocols by invoking the appropriate verb.

Given that a full-duplex conversation can be emulated with two half-duplex conversations, why should APPC platforms be enhanced to support the option set for full-duplex conversations?

The full-duplex conversation option set should be used instead of the half-duplex accommodation provided by CPI-C because network resources are used more effectively. To support two conversations, the network must activate two sessions. Each session requires memory for control blocks, memory for buffer pools, administration overhead (activation, list searching, process switching, network management session awareness), and a session address. By using the full-duplex conversation option set, this overhead is cut in half.

Furthermore, some APPC platforms, such as IBM's Customer Information Control System (CICS), cannot attach the server's conversation back to the originating client program. These platforms always activate a new program process when any new conversation attach request arrives. By using the full-duplex conversation option set, the client creates both the send and receive paths by allocating a single conversation to the server. In this case, even these platforms can support full-duplex applications.

Half duplex has two primary disadvantages. First, each simultaneous transaction requires a duplicate set of network resources. Second, while the application is actively processing the transaction, the network resources for that transaction are idle.

Full Duplex. With full-duplex protocols, the application takes on an additional burden to implement the primitives. If the primitive is used by the application, the application must format special records to implement the necessary protocols. Subfunction identification, race resolution, synchronization, coordination, and correlation must be handled by the application. The communication subsystem is unable to help because it does not understand the relationship between the records. It cannot tell

Will full-duplex conversations be used to implement Multiprotocol Transport Networking?

IBM's networking blueprint proposes a world where you no longer need to buy a network based on the application interface used by your favorite programs. For example, if your favorite program uses TCP sockets, it can use an SNA network as easily as a TCP/IP network; if your favorite program uses APPC, it can use a TCP/IP network as easily as an SNA network.

For the networking blueprint, this capability is based on Multiprotocol Transport Networking (MPTN), which IBM has proposed to X/Open to become a standard. The core of this MPTN technology is a set of carefully crafted compensators—network programs that compensate for network differences and then envelop protocols for transport through the target network. A compensator is a protocol transport application. A compensator that transports a protocol through an SNA network uses full-duplex APPC conversations.

This is good news for software vendors who want to make a profit from developing compensators. These vendors will be able to utilize an open application interface for full-duplex APPC conversations. They will not need to use an undocumented interface hidden in the middle of the SNA protocol stack.

which application subfunction detected an error, which incoming record to purge, which application subfunction is waiting for confirmation, or which application subfunction should be allowed to continue sending while waiting for a confirmation.

The primary advantage of a full-duplex conversation is that the application can bundle independent pieces of work onto a single set of network resources. Even if one subfunction is idle, other subfunctions can continue to use the resources.

Choosing: Half Duplex or Full Duplex

How would you, as an application designer, decide between a half-duplex and full-duplex APPC conversations? There are three simple guidelines:

1. If your application is similar to one of the full-duplex application models discussed above, use full-duplex conversations.
2. If you have an existing application that uses a full-duplex service, like TCP sockets or NetBIOS sessions, and you want to enhance that application to use APPC, use full-duplex conversations. If the application is really a half-duplex application, you have already coded the logic required to perform the functions built into APPC. It will be easier to use that same logic with full-duplex conversations than to remove the logic and use the features built into APPC half-duplex conversations instead.
3. Otherwise, use half-duplex conversations and use the powerful APPC features your application requires.

Programming Considerations

CPI-C provides a consistent application programming interface for applications that require program-to-program communication. CPI-C Level 2, currently in the proposal stage, contains enhancements for full-duplex conversations. This discussion of programming considerations uses calls and terms taken from that proposal.

Availability of Full-Duplex Support

Full-duplex conversations are not possible unless the APPC platforms on both the client's computer and the server's computer support full-duplex conversations. Before you start developing an application that uses full-duplex APPC conversations, make sure that full-duplex support is available on all the platforms where your application will run.

Full-duplex support will become available in stages. In the first stage, CPI-C Level 2 uses two half-duplex conversations to provide the appearance of a full-duplex conversation interface to applications. (See the sidebar on page 6 for more information.) In the second stage, APPC protocols will be enhanced (APPC option set 112) so a single conversation supports the full-duplex conversation interface.

SNA Perspective expects the first stage to appear in workstation products in the second half of 1993 and the second stage to appear in workstations, midrange, and mainframe systems by the end of 1994. If you intend to implement full-duplex applications, now is the time to inform your APPC platform vendors of your requirements and to encourage them to implement the APPC full-duplex option set of CPI-C Level 2.

Configuration Considerations

An APPC platform that supports full-duplex conversations requires no special configuration to use them. There are no new configuration parameters on mode definitions, logical unit (LU) definitions, or partner LU definitions.

Furthermore, on such a platform, any given APPC session can serially support full-duplex and half-duplex conversations. A free session can be assigned to either type of conversation.

Full-duplex applications can thus be smoothly integrated into existing networks. The only configuration consideration is whether or not the session limits must be increased to support the new application. The following example describes this capability.

1. A client allocates a half-duplex conversation to a particular target with a particular mode and APPC assigns the conversation to a particular session.
2. When that half-duplex conversation terminates, that session is returned to the free pool.
3. A client (the same or a different client) allocates a full-duplex conversation to the same target with the same mode. The conversation is assigned to the same session.
4. When the full-duplex conversation terminates, the session is again returned to the free pool. The session can be used over and over, with either type of conversation using it.

Will full-duplex conversations allow APPC programs to communicate directly with TCP socket programs?

You may be tempted to believe that full-duplex APPC programs can communicate directly with TCP socket programs. Not only is this not true, it is also a questionable goal.

Programs in a distributed application must cooperate at every level. A client and server must agree on the content of the data, the organization of the data, the order of the data flows, and the state information exchanged. If the client and server attempted to communicate using different interfaces (interfaces with different semantics), cooperation would be impossible.

If an APPC program attempts to communicate with a TCP socket program, either the TCP socket program needs to include a compensator to give it the semantic equivalent of APPC conversations, or the APPC program needs a compensator to give it the semantic equivalent of TCP sockets. In either case, the end result is like-to-like communication.

A better strategy is to let the programs communicate using a standard application interface for conversations, remote procedure calls, message queues, or some other program-to-program communication facility. Then, when necessary, place the compensators inside the network so that the programs are network independent.

Multiple Threads Versus Nonblocking Calls

A full-duplex conversation would do little good if a program is unable to send and receive at the same time. But if a program is suspended while waiting for a receive operation to complete, it is unable to send. Likewise, if the program is suspended while

waiting for a send operation to complete, it cannot receive. To allow multiple operations to be outstanding at the same time, full-duplex conversations must support programs with multiple threads and provide nonblocking calls.

Some systems permit a program to have multiple threads. Each thread executes instructions independently and communicates with other threads using shared memory, semaphores, queues, or other shared objects. A thread may be suspended while waiting for a call to return, but other threads continue to work.

Some systems permit a program to associate a wait object with a call. When the call cannot be completed immediately, the call is not blocked. Instead, control is returned to the program with an indicator that the operation is incomplete. The program continues with other work and occasionally checks the wait object to see whether the operation has finished. When the operation is complete, the program continues with the required processing.

One common design for full-duplex applications creates two threads: one thread performs send operations and the other thread performs receive operations. When the receive thread is blocked because the partner program has not yet sent data, the send thread can continue to send data. When the send thread is blocked because the partner program has not received enough data to free the required network buffers, the receive thread can continue to receive data.

Another common design for full-duplex applications uses nonblocking calls—each call is issued with a wait object. When a call returns with an incomplete indicator, the program places its wait object in a list containing wait objects for all incomplete calls. When there is no work to do, the program waits on the list. When a wait object is posted, the program performs the required processing, issues another nonblocking call if necessary, and continues the loop.

Allocating and Accepting a Conversation

When allocating a full-duplex conversation, the client program must specify that a full-duplex conversation is required during the Allocate call. Full-duplex conversations do not support confirmation requests, so a full-duplex conversation is always allocated with a synchronization level of NONE.

Likewise, the server program must be defined with matching capabilities and use an Accept_Conversation call to accept a full-duplex conversation with a synchronization level of NONE.

Deallocating a Conversation

For normal termination, the conversation is not deallocated until both programs make a Deallocate call. When a program finishes sending data, it must make a Deallocate call to flush the final record and inform the partner. Then, if the program hasn't already received a return code indicating that the partner has deallocated, it must continue to receive until it does.

Either program can make the Deallocate call first and the programs can make the Deallocate calls one after the other or at the same time. Also, there is no limit to the amount of data a program can send after being informed that the partner has deallocated. In these situations, you are free to make your own rules when you design your application.

For abnormal termination, a single Deallocate call with a deallocate type of ABEND is enough to deallocate the conversation. After this call, neither program is permitted to send, a return code informs the partner program that it cannot send, and both programs must continue to receive until a return code informs them that the conversation has been abnormally terminated.

Sending and Receiving Data

After allocating a full-duplex conversation, the client program can make Send_Data calls and Receive calls at any time. Likewise after accepting a full-duplex conversation, the server program can make Send_Data calls and Receive calls at any time.

As for half-duplex conversations, a Send_Data call does not cause the data to be transmitted to the partner immediately. To reduce overhead, the data will

be buffered until the buffer overflows or the program forces the data to be transmitted. Unlike half-duplex conversations, a Receive call does not force the data to be transmitted. Furthermore, full-duplex conversations do not support synchronization calls that force the data to be transmitted.

In a full-duplex conversation, the program can force the data to be transmitted by using a Flush call when there is more data to send or by deallocating the conversation when the program has finished sending. Of course, to reduce overhead, the flush function can be combined with the send function by using a Send_Data call with a send data type of Flush.

Sending and Receiving Expedited Data

Some full-duplex applications need to send expedited data—urgent data that must pass all the normal data previously sent. For example, a record that cancels a previous request, sometimes called a forward abort, would use this feature. Full-duplex conversations support special calls for this purpose: Send_Expedited_Data and Receive_Expedited_Data.

As long as the conversation has not been deallocated, a program may send from 1 to 86 bytes of expedited data at any time. After sending expedited data, the program may send more expedited data, but the Send_Expedited_Data call may be suspended if the partner program has not received the previous expedited data.

When the partner program sends expedited data to the program, status indicators on every Receive, Send_Data, and Send_Expedited_Data call will inform the program to receive expedited data until the expedited data has been received. If your program is not making one of these calls, it cannot be informed. So if the program is designed to use expedited data and expedited data may arrive while the program is not making calls, a Receive_Expedited_Data call should be kept active at all times. To keep the Receive_Expedited_Data call from interfering with normal data processing, either it should be placed in its own thread or a non-blocking verb should be used and the wait object should be checked frequently.

Conclusions

Although half-duplex APPC conversations are suitable for most distributed applications, full-duplex conversations are a welcome enhancement. Full-duplex applications can easily be implemented with full-duplex conversations and your programs can still benefit from the reliability and built-in services provided by APPC conversations. CPI-C Level 2 will provide a consistent interface for full-duplex conversations on APPC platforms.

References

Bader, L. D. and Walker II, J. Q., "Classic Client-Server Transactions Using APPC." *IBM Personal Systems Developer,* Spring 1991.

Systems Application Architecture Common Programming Interface Communications Reference. IBM Document Number SC26-4399, June 1992.

Gray, J. P., Hansen, P., Homan, P. Lerner, M. and Pozefsky, M., "Advanced Program-to-Program Communication in SNA." *IBM Systems Journal,* Vol. 22, No. 4, 1983.

Systems Network Architecture LU 6.2 Reference: Peer Protocols. IBM Document Number SC31-6808, September 1990.

Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2. IBM Document Number GC30-3084, September 1991. ■

(continued from page 1)

HPR Components

HPR consists essentially of two parts—a transport protocol called Rapid Transport Protocol (RTP) and a routing protocol called Automatic Network Routing (ANR).

APPN now has two routing modes—HPR and ISR. An illustration of the difference between HPR and ISR is shown in Figure 6. (Technically, IBM architects might represent RTP and ANR as operating at an enhanced layer 2 rather than at layers 3 and 4.) Both HPR and ISR use the same APPN control point (CP), the same topology protocols, and the same directory services. HPR can coexist with APPN's current ISR in APPN nodes and in APPN networks. The mode being used is transparent to the upper layers.

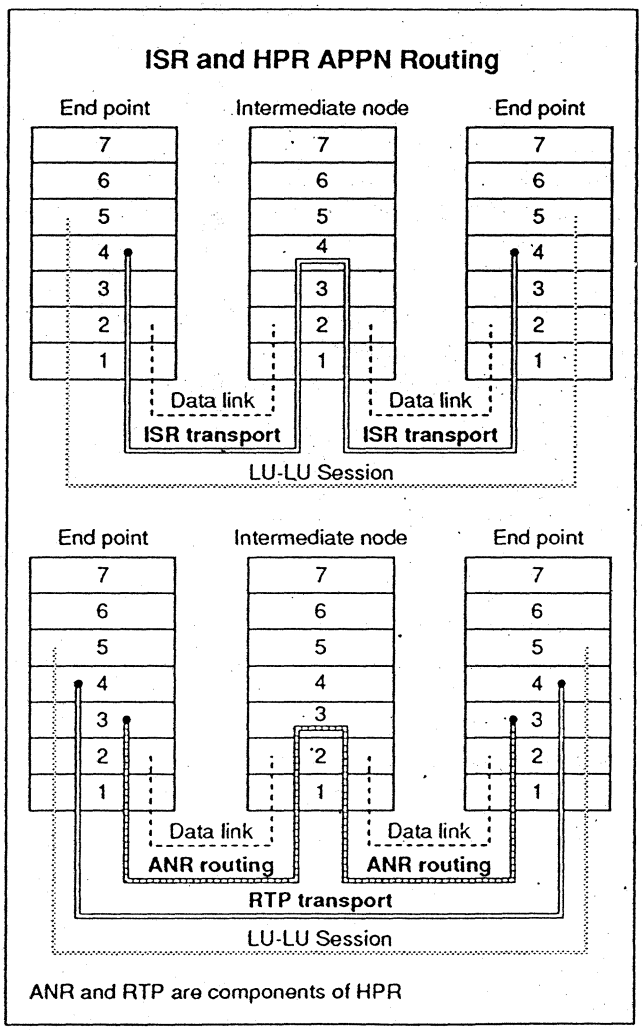

Figure 6

Before discussing HPR in more detail, we briefly examine four basic routing techniques and then review ISR.

Four Routing Techniques

Four basic routing techniques are widely used—virtual/explicit routes, label swapping, destination routing, and source routing. They are of interest here because they are used, respectively, by subarea SNA, ISR, TCP/IP, and HPR.

Virtual Routes and Explicit Routes

Subarea SNA is often referred to as a "nonroutable" protocol. This is because the paths are preconfigured and the primary routing decisions for each session are made at a centralized host rather than at any node in the network. Furthermore, the traffic between a peripheral node and a subarea node does not include a full network address but includes instead a name or local address that can only be understood by subarea nodes.

Explicit routes and virtual routes, which are preprogrammed into VTAM, define the possible combinations of links between any two subarea nodes in the network. For each session, VTAM assigns an explicit route and virtual route based on the session characteristics and requirements. The explicit route number is used by each subarea node (VTAM or NCP) to select the next hop from its preprogrammed table. Virtual/explicit routing uses some qualities of source routing at the host node and some qualites of destination routing at the intermediate nodes.

Label Swapping

Several protocols, such as Asynchronous Transfer Mode (ATM) and APPN's ISR, use label swapping. In this discussion, we will use ISR as an example. The routing is done hop by hop across a path set up at the beginning of the session. Each session has a unique fully qualified procedure correlation identifier (FQPCID). The initial connecting message (BIND) informs each intermediate node of where next to send each packet coming in on that session.

Between each set of two nodes, the traffic for a particular session uses a local form session identifier (LFSID) to distinguish it from other traffic using the same link. Each pair of intermediate nodes assigns an LFSID to each session using that hop along its path. These LFSIDs are stored in each node with the associated FQPCID in a table. For the duration of the session, each packet that comes in from either direction has its LFSID stripped off and the appropriate LFSID for that session is added on for the next hop.

Destination Routing

Destination routing is used by several protocols, including the Internet Protocol (IP). In this discussion, we will use IP as an example. In destination routing, the destination address is carried in the front of the packet and is used for making routing decisions. Each routing node that sees the packet has one of several ways, such as a cache, preprogrammed table, or filter, to tell it whether to leave the packet on its current network or forward it over one of the outbound links.

If the packet is to be forwarded and more than one link could lead to the destination, the routing node uses one of several procedures, such as Open Shortest Path First (OSPF), to select the next hop. OSPF and other routing topology protocols exchange routing table updates frequently across the network. Destination routing is connectionless so each packet could arrive at the destination by a different path and in a different order, depending on link availability and congestion.

Source Routing

Source routing is used by token ring bridging at layer 2, by APPN's HPR/ANR at layer 3, and in some high-speed trials such as the Aurora test bed. In source routing, the source node makes the routing decisions. In this discussion, we will use HPR as an example.

The source node, such as an APPN network node server, uses one of several means, such as a cache or a locate request, to determine the location of the target application. From its topology database, it then calculates the optimal route to the target node for the desired class of service. Each packet header includes information for every hop along the route in the header. Each intermediate node examines the header, removes its own label, notices the next link indicated in the header, and forwards the packet on that link. The 1–2 byte label for each hop has local significance only—a table at the intermediate node indicates how to interpret the label.

If a link or node on the selected route goes down, the source node obtains another route without disrupting the session and sends packets with the new route hops listed in the header.

ISR Review

ISR is a component of APPN, as shown in Figure 7, and is itself a set of components. ISR's functions include error recovery, adaptive pacing, and segmentation and reassembly. ISR includes the session connector in an intermediate node and the LU half-session in an endpoint node, which are functionally analogous. ISR's label swapping, FQPCIDs, and LFSIDs are discussed above under "Label Swapping." Although architecturally APPN is located above the SNA path control layer, as illustrated in Figure 7, readers should note that some

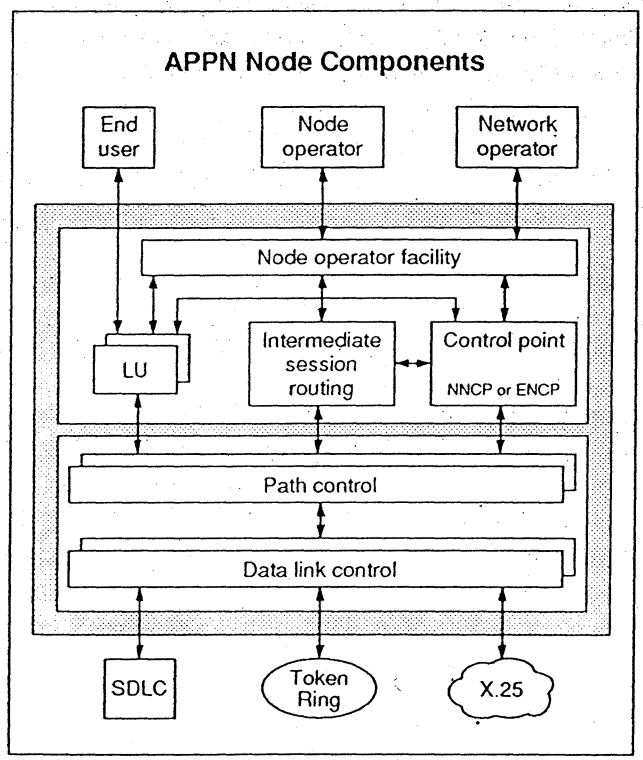

Figure 7

discussions of APPN today, particularly in internetworking circles, include path control functions under the term APPN.

In internetworking terminology, ISR serves approximately the same functions as TCP and IP. Many users think of APPN as a whole being comparable to TCP/IP. However, in addition to ISR, the APPN architecture includes a topology protocol, which serves a similar function as OSPF or Cisco's IGRP. It also includes a distributed directory services component, which serves a function similar to the domain name server in TCP/IP.

Both subarea SNA and APPN with ISR are connection-oriented at layer 2 (data link) and every layer above it. They also perform error recovery at layer 2, layer 4 (transport), and layer 5 (flow control). ISR also performs transport-level processing at each intermediate node, including segmenting/reassembly, pacing, and priority queuing. This level of robustness is appropriate for environments with slow or unreliable physical networks.

However, ISR ensures a level of reliability beyond the needs of many environments today, given the increasing reliability of underlying physical networks such as LANs and T1 links. HPR was developed to address some of these concerns. As discussed below, HPR will allow intermediate APPN nodes to operate at level 3 using a connectionless network layer protocol, which will decrease processing overhead and storage requirements and increase performance.

Connection Networks

As a complement to ISR, IBM offers a capability called connection networks or the virtual routing node. This allows nodes to connect to each other over a LAN or an internetwork. The connection network feature is another way, besides HPR, to get around ISR hop-by-hop routing. See the sidebar on page 16, "Meanwhile...Connection Networks."

High-Performance Routing

HPR is a set of new transport/routing protocols that can be used instead of or in conjunction with ISR. IBM indicates that products implementing HPR should be available in twelve to eighteen months:

Because of the ISR/HPR boundary function provided with HPR, a session can have some ISR segments and some HPR segments. *SNA Perspective* expects that, for many years, all APPN implementations with HPR will still include ISR.

As shown in the left of Figure 8 where two adjacent links of an APPN session are HPR-capable, HPR can be used for that segment while ISR is used for the other links. If only one link is HPR-capable, as shown in the right of the figure, APPN could still use HPR but it would offer no additional benefit. A node indicates its capability for HPR in its topology database update (TDU).

Architecturally, HPR could be added in software to an existing APPN/ISR node with no hardware changes, although performance will increase significantly more if the hardware is also adapted.

Automatic Network Routing

The network layer of HPR is ANR. Where HPR is used, ANR functions are performed at every node. ANR provides connectionless, stateless source routing. It services the outbound link based on priority and may discard incoming packets in the event of congestion.

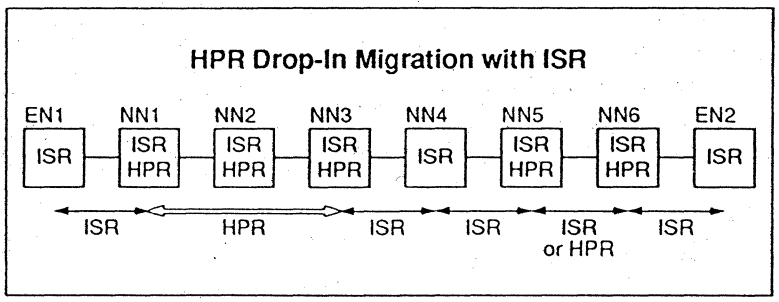

Figure 8

Readers may see ANR described elsewhere as a connection-oriented network protocol with several connectionless services. Although ANR has qualities of both connectionless and connection-oriented routing, we believe it is most appropriately characterized as connectionless.

Architecturally, ANR could run on top of a variety of connectionless data links, such as IEEE 802.2 type 1, HDLC Point-to-Point protocol (PPP), and frame relay, as well as over connection-oriented data links, such as SDLC or IEEE 802.2 type 2. *SNA Perspective* expects IBM to first implement HPR on token ring and frame relay.

Rapid Transport Protocol

The connection-oriented transport layer of HPR is RTP. It performs the following functions: connection awareness, reliable delivery, reordering, packet sizing, flow control/congestion control, nondisruptive route switching, and session multiplexing. RTP is only used at the endpoints of an HPR session.

Adaptive Rate–Based Flow Control

IBM has also added a new technique for flow control and congestion control for HPR. This technique is called adaptive rate–based (ARB) congestion control. ARB is part of RTP and is only used at the endpoints of an RTP connection. APPN's existing adaptive session-level pacing is still used with any ISR links in the route.

HPR Benefits

Drop-In Migration. HPR can coexist with ISR through drop-in migration. HPR-capable nodes can exist in a route alongside ISR-only nodes. This may be possible with a software-only upgrade, though performance improvements would be limited. *SNA Perspective* expects that all network nodes will continue to include ISR and they will, in addition, include HPR over time.

Connectionless Routing. HPR/ANR is not connection-oriented for each hop as ISR is and the intermediate nodes only process packets up to layer 3, so less processing and storage is required at each intermediate node. This also allows nondisruptive adaptive rerouting in case of failed nodes or links.

Improved Performance. IBM claims that HPR can offer significant improvements over ISR with a software-only upgrade and major improvements in combination with hardware optimization.

Reduced Storage. Since HPR does not need to process each packet up to the transport level at each intermediate node using its session connector tables or maintain control blocks to store pacing and error-recovery data, less storage is required at each node. In addition, if the end node has HPR, its network node server does not need to store as much information to support that end node.

Congestion Control. HPR offers congestion avoidance and flow control through adaptive rate–based congestion control.

Still Missing

Multimedia. Some IBM sources and analysts are portraying HPR, to some degree, as a transport for multimedia applications. However, HPR does not allocate bandwidth. Therefore, while it is suited for several client-server multimedia applications, it does not have support for the isochronous traffic typical of desktop videoconferencing and the real-time video portion of some multimedia applications. *SNA Perspective* expects that gigabit APPN, expected in 1995, will offer this support.

Multiprotocol. IBM has also not indicated if other protocols will be able to run directly over either HPR as a whole or over ANR separately. IBM's Multiprotocol Transport Networking (MPTN), which will be addressed in a future *SNA Perspective* article, presents a standard way to have applications expecting one transport type to be run over another transport type. These mixed transport types include TCP/IP, SNA/APPN, NetBIOS, and OSI. IBM has not discussed how HPR fits into the MPTN picture. *SNA Perspective* believes that HPR could replace ISR in any MPTN configuration.

In the Meanwhile...Connection Networks

Connection networks were developed for APPN in the late 1980s for two reasons. First, it became clear that, over high-speed reliable LAN links, ISR's hop-by-hop robustness was unnecessary. Second, ISR did not deal as efficiently with LAN and internetwork topologies where a node could be logically adjacent to a large number of other nodes. ISR required a large number of topology definitions in such an environment.

The connection networks feature allows an APPN user to define a LAN or internetwork of any size or complexity as a single connection network. This connection network allows an APPN network node to treat this connection network as a single virtual routing node, appearing as a single APPN/ISR hop with zero intermediate nodes. Two APPN end nodes which are both on the same connection network are treated by APPN as if they were adjacent nodes. The actual LAN or internetwork connection between them is transparent to APPN.

An example of a connection network is shown in Figure 9. Because IBM has only implemented connection networks on token ring LANs and over data link switching, the example uses token ring LANs. However, connection networks could be implemented over Ethernet, X.25, frame relay, SMDS, ATM, and other environments.

1. The user in this example has defined the internetwork which consists of LAN A, LAN B, and LAN C as Connection Network 1. Any end node can be defined for one or more connection networks.

2. The two network node servers in this example are not connected over the internetwork but through a separate SNA network. However, they could be connected over the internetwork using APPN over Data Link Switching or APPN over sockets—both these features are available on the IBM 6611 router and will be available for use with the licensed network node code. Either way, the network nodes serving the end nodes for Connnection Network 1 must be able to communicate with each other using APPN without connection networks.

3. The user configures each APPN end node on LAN A, LAN B, and LAN C to be adjacent to Connection Network 1. The user does *not* configure each APPN end node to be adjacent to every other APPN end node available on this internetwork. (Without connection networks, this second method is the usual APPN procedure.)

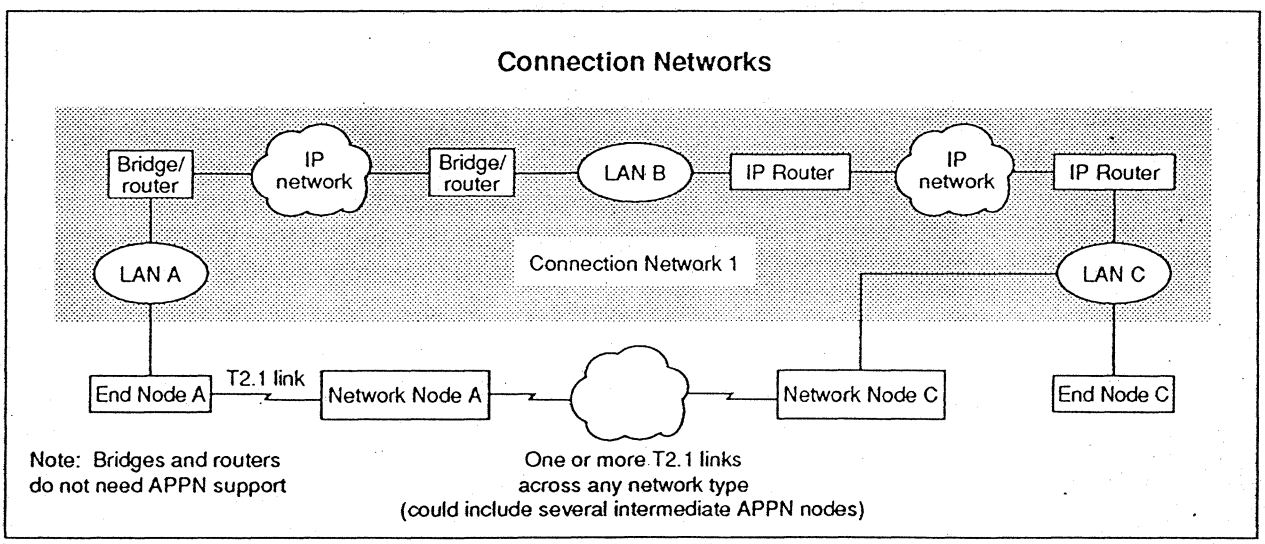

Figure 9

In the Meanwhile...Connection Networks (continued)

4. When it is powered on, each APPN network node finds every adjacent network node, registers its topology to it, and receives information from it. The APPN network node then establishes a control point-control point (CP-CP) session with one or more adjacent network nodes. These are normal procedures for all APPN network nodes regardless of whether they will use connection networks.

5. When it is powered on, each APPN end node finds its network node server and may also register its resources.

6. End node A on Connection Network 1 requests a session with a certain application, not knowing where that application is located. Assume the target application is in end node C.

7. At end node A's request, its network node server, network node A, searches for the application with the other network node servers using the standard APPN locate procedures.

8. Network node C responds with end node C's tail vectors—a list of its links including Connection Network 1, and includes its medium access control (MAC) and service access point (SAP) address.

9. Network node A sees that both end nodes are on Connection Network 1 and, weighing all other possible links, selects the route with only one hop—Connection Network 1.

10. Network node A tells end node A to set up its own session through its apparently adjacent link to Connection Network 1. Without connection networks, the usual APPN procedure would be for the data to go from end node A to network node A, then across an optimal APPN path to network node C, and then to end node C.

11. End node A uses the usual token ring source route bridging discovery procedure to find the node with the given MAC address. In this example, the search would be broadcast across the LAN internet and perhaps some of the SNA network. This step would not be necessary in an APPN process without connection networks.

12. End node A then sets up an APPN link with end node C through the usual XID process and BIND. Without connection networks, network node A would only need to send a BIND to start the session. The APPN end nodes believe the link is adjacent. Any actual bridges or routers in Connection Network 1 are transparent to APPN. The APPN traffic is equally transparent to the bridges and routers; it can be sent as any other token ring traffic.

This process seems cumbersome at first—and it is—but as caches are developed it becomes increasingly simplified. Also, the session setup effort is more than offset by the reduced predefinitions (especially if nodes are frequently added or moved) and the subsequent efficiency. One benefit of connection networks is that an intervening IP router or router internetwork can be used efficiently that could not be used by APPN. The internetwork can, alternatively, be used by APPN if all the intermediate routers have APPN installed or if the routers at each end have support for APPN over DLS or APPN over sockets.

A caveat: it is recommended that an internetwork defined as a connection network should have relatively high and consistent performance characteristics. Otherwise, for example, if two LANs with a satellite link were included in a single connection network, an APPN network node would treat the connection network as one APPN hop and would choose that route, blind to the significant satellite delay, even if a faster multihop terrestrial APPN route were available. ■

Current Concerns about APPN

Nine concerns about APPN are listed in Table 1. The first three are directly addressed by HPR. IBM has stated that several others will be addressed in future releases of APPN that have features unrelated to HPR and/or future releases of VTAM or NetView.

High Network Node Storage Requirement

With ISR, each intermediate network node must maintain a session connector control block for each active session as well as the network topology. HPR will include the full routing information in each packet so these session routing tables do not need to be stored on each intermediate node.

High Network Node Processing Overhead

With ISR, each packet that enters a network node is stripped of its local form session identifier (LFSID) and given another LFSID for the next hop on the path. This involves protocol processing and a header change on every packet. HPR will include the full routing information in each packet, eliminating some processing overhead at the intermediate nodes. Also, intermediate nodes using HPR are not performing error recovery, pacing, or segmentation/reassembly, which also decreases overhead.

No Dynamic Rerouting Upon Failure

ISR fixes a path for each session, using the topology database and required class of service to calculate the optimal route at session initiation. All traffic for that session runs over that path. If any link in the path fails, the session is lost and must be restarted. The network software may be written to hide this reinitiation from the end user, but it can cause significant delays.

HPR is able to discover a path failure and switch to a new path without losing the session. Also, some data links like X.25 and frame relay can nondisruptively reroute around failures without APPN's awareness.

No Adaptive Rerouting Upon Congestion

With both ISR and HPR, the path is fixed for the duration of the session unless that path is lost because of a link failure at some point. If a link becomes congested, neither ISR nor HPR selects another route. (However, if a link becomes extremely congested, an HPR node may interpret a significant delay as a failed link or node and reroute.)

Some protocols like TCP/IP theoretically allow each packet to proceed by the best available path, routing around congestion, although in practice this capability is not implemented by any TCP/IP vendor.

On the other hand, ISR and HPR both focus on congestion and flow control, although in different ways. Rather than routing around congestion, the APPN approach for both modes is to prevent congestion by controlling the traffic pace or rate.

SNA Perspective considers the debate about rerouting around congestion versus preventive congestion control to be a philosophical issue. Each side has its advantages, but the difference has minimal impact on users in real-world situations.

Multiple APPN Networks

APPN networks can be separated into subnetworks through the use of different NETIDs. An APPN node that allows traffic to pass between two or more subnetworks is called a border node. An interchange node, on

Current Concerns about APPN

Concern	IBM Plans to Address
High network node storage requirement	HPR
High network node processing overhead	HPR
No dynamic rerouting upon failure	HPR
No adaptive rerouting upon congestion	No
Limited multiple APPN networks (border node)	Future—VTAM, network node
No multilink APPN transmission groups	Expected, but no formal IBM statement
Limited dependent LU support	Current—via VTAM, future—any NN
Little network management by NetView	Current—SNMP, future—more NetView
Proprietary to IBM	Much more open in '92, even more expected in '93

Table 1

the other hand, is a node that allows traffic to pass between an SNA subarea and an APPN network. Both border node and interchange node capabilities can be added to a network node. HPR is completely independent of either border node or interchange node features.

Currently, border node is only implemented on the AS/400. The AS/400 border node only allows traffic to pass between two adjacent subnetworks. Today, traffic cannot pass from one APPN subnetwork through an intermediate APPN subnetwork to a third APPN subnetwork.

IBM made a statement of direction in March 1992 that a future release of VTAM will include border node. The VTAM border node will be enhanced to allow any number of APPN subnetworks to be traversed. *SNA Perspective* expects that border node will appear in VTAM 4.2, which we believe will be announced sometime later in 1993 after VTAM 4.1 begins shipping in June, and could start shipping early in 1994.

IBM has also indicated that border node may be offered as an option with a future release of the licensed APPN network node code. *SNA Perspective* does not expect this to be available until after it is provided on VTAM, though it could be available sometime in 1994.

Multilink Transmission Groups

Transmission groups in subarea SNA allow users to combine several parallel links between two nodes in a way that allows them to appear as one link and adjust traffic between them in cases of congestion or link failure.

Currently, an APPN transmission "group" can only consist of one link. Until VTAM 4.1, subarea SNA links over token ring and frame relay were similarly limited to one link per group; only SDLC links could be combined. IBM has indicated, though not formally, that it intends to support multilink transmission groups for APPN in a future release of VTAM. *SNA Perspective* expects that multilink APPN transmission groups will also be included in a future release of the licensed network node code but probably not until 1995.

Because of certain efficiencies of subarea SNA transmission groups combined with user frustration with the current limitations of APPN transmission groups, IBM made a statement of direction in September 1992 that a future release of VTAM will support APPN sessions running over subarea SNA links using subarea virtual routes and explicit routes.

HPR does not directly address the transmission group issue. However, HPR allows adaptive rerouting on link failure, which is an alternate way of providing one of the benefits of multilink transmission groups—switching to an alternate link. On the other hand, as discussed above, initial HPR products are *not* expected to provide adaptive rerouting for congestion control. Multilink APPN transmission groups will benefit users who would like this feature because it will allow them to balance the load across multiple links and permit addition of incremental bandwidth.

Dependent LU Support

Support for existing dependent LU devices and applications is essential for subarea SNA users migrating to APPN. The November 1992 issue of *SNA Perspective* discussed in depth several current and forthcoming capabilities from IBM and other vendors that address this issue including the Dependent LU Server/Requester and various encapsulation techniques. Dependent LU support is not directly affected by HPR. APPN nodes with either HPR or ISR will be equally able to support dependent LU traffic. This support can be either as access nodes (at the periphery of the APPN network) with one of these capabilities or as intermediate nodes without any additional capability.

Little Network Management by NetView

The APPN management services architecture, based on the OSI Common Management Information Protocol (CMIP), is quite impressive. But few of these architected APPN support features have been implemented in NetView. APPN network management issues were discussed in the February and April 1992 issues of *SNA Perspective*. Because of both this limitation and the popularity of the Simple Network Management Protocol (SNMP) in the internetworking community, IBM is including an

SNMP agent in the first release of the licensed APPN network node. The SNMP support is provided along with the SNA Management Services (SNA/MS) capabilities present in all current IBM APPN offerings, which allow some management by NetView. APPN network management is not directly affected by the presence or absence of HPR. However, NetView will need some additional code to manage HPR.

Proprietary to IBM

IBM made several meaningful moves to open APPN during 1992, including licensing the network node source code and agreeing to publish the network node specifications, both of which should be available in March 1993. IBM is also moving to involve other vendors in APPN implementer workshops which will also serve as forums for feedback on future APPN developments.

However, even with this significant openness, APPN is still in several ways a proprietary protocol—IBM owns it, defends its patents, controls its development, and licenses its use. This is not to say that proprietary is by definition "bad" or that a proprietary product cannot succeed in the market. Novell's very successful NetWare is proprietary, as is Cisco's very successful IGRP. A significant difference between these two and APPN is that NetWare and IGRP were early entrants in their respective markets when there was no formal and formidable competition.

APPN, on the other hand, is breaking onto the internetworking scene (six years after its first implementation) in the face of TCP/IP, which has a major market share and is significantly standardized, very inexpensive, and widely implemented. A large percentage of the networking workforce, and students too, are experienced in using TCP/IP, while subarea SNA experience is less common and APPN expertise is almost nonexistent.

These are significant obstacles for APPN to overcome even with its many technical advantages. All these topics will be addressed in a future issue of *SNA Perspective* which will compare APPN and TCP/IP as alternative migration paths for current subarea SNA users.

Conclusions

APPN continues to evolve and HPR is one piece of that evolution. HPR appears to provide several advantages over ISR, particularly in intermediate node performance and storage. It also seems more interoperable with other architectures such as TCP/IP.

The advantages of HPR are probably a least a year to eighteen months away for product implementations, although beta tests might start before the end of 1993. All current APPN implementations, including LEN nodes, are based on the ISR technology. ISR is part of the network node licensed code and is documented in the network node specification, both of which are scheduled to be available in March. *SNA Perspective* expects that the next release of the APPN licensed network node code will include HPR and will probably ship by the first quarter of 1994.

Until HPR is available, APPN/ISR users with LANs and internetworks can take advantage of the capabilities of connection networks to improve performance and allow a degree of alternate routing.

HPR is not the final word in APPN routing. IBM has also been discussing gigabit APPN, a form of APPN that is being optimized for the emerging very high-speed cell networks. It is expected to be compatible with ATM. Gigabit APPN is expected to be formally announced in 1994 and available in 1995. ■